

University of Warwick institutional repository: <http://go.warwick.ac.uk/wrap>

A Thesis Submitted for the Degree of PhD at the University of Warwick

<http://go.warwick.ac.uk/wrap/65186>

This thesis is made available online and is protected by original copyright.

Please scroll down to view the document itself.

Please refer to the repository record for this item for information to help you to cite it. Our policy information is available from the repository home page.

Design for Robustness of Complex Automotive Electronic Systems

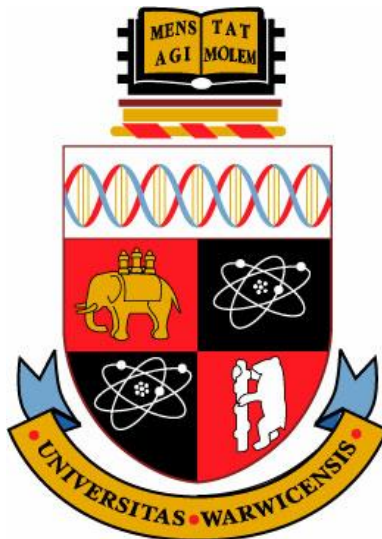
by

Ross McMurran

A thesis submitted in partial fulfilment of the requirements for
the degree of Doctor of Philosophy in Engineering

School of Engineering

University of Warwick



Sept 2014

This thesis is dedicated to the memory of my father
George Crossan McMurran.

Contents

Contents	i
List of Figures	vi
List of Tables	x
Acknowledgements.....	xi
Declarations	xii
Abstract	xiii
List of Abbreviations& Terminology.....	xv
Chapter 1 – Introduction	1
Chapter 2 – Literature Review	6
2.1 Introduction	6
2.2 Robustness.....	6
2.3 Complex Systems of Systems	14
2.4 Automotive Electronic Systems	16
2.4.1 Complexity of Automotive Electronic Systems	16
2.4.2 Impact of Increasing Complexity of Automotive Electronic Systems	18
2.4.3 Existing approaches to managing Complexity of Automotive Electronic Systems	21
2.5 Agile Software Development Methods	26
2.6 Quality methods	27
2.6.1 Failure Mode Avoidance (FMA).....	27
2.6.2 Design for Six Sigma	29
2.7 Design Methods for Robust Systems	30
2.7.1 Design principles for complex systems	30
2.7.2 Fault tolerance design approaches	31
2.7.3 Fault recovery approaches	34
2.7.4 Self Stabilization.....	35
2.8 Safety engineering.....	36
2.8.1 Safety Standards.....	36

2.8.2	Safety cases	41
2.9	Modelling Complex Systems	43
2.9.1	Architectural Modelling Approaches.....	44
2.9.2	Co-Simulation Modelling Approaches	50
2.10	Robustness Test Methods	53
2.11	Formal methods	59
2.12	Discussion.....	63
2.13	Conclusions	67
Chapter 3	Case Study Review of Automotive Electronics Robustness Issues	68
3.1	Introduction	68
3.2	Case Study Methodology	69
3.2.1	Case Study Questions.....	69
3.2.2	Selection of Cases to Review.....	70
3.2.3	Analysis Method	71
3.3	Case Study Results and Analysis	78
3.3.1	Overall Occurrence of Robustness Issues.....	78
3.3.2	Impact of robustness issues.....	78
3.3.3	Fault Categorization.....	79
3.3.4	Robustness Issues – Phenomenological Fault Causes	81
3.3.5	Activation Patterns.....	84
3.3.6	Resulting Actions.....	86
3.3.7	Earliest Potential Detection Point?	88
3.4	Threats to validity of study.....	91
3.4.1	Selection of cases.....	91
3.4.2	Selection of the categories	92
3.4.3	Categorisation Decisions	92
3.5	Conclusions	92
3.5.1	Overview of Results.....	92

3.5.2	Discussion on Implications for Robustness Framework.....	94
Chapter 4 -	Development of Design for Robustness Framework	97
4.1	Introduction	97
4.2	Design for Robustness Framework Requirements	97
4.3	Overall design for robustness framework	99
4.4	Design for Robustness Framework Elements	101
4.4.7	Data Analysis	107
4.5	Conclusions and Next Steps.....	108
Chapter 5 –	Development of New Robustness Techniques through Application to an	
Infotainment System	110
5.1	Introduction	110
5.2	MOST Infotainment Systems.....	110
5.3	Infotainment Robustness Case Development.....	113
5.4	Robustness Modelling.....	124
5.4.1	Approach To Modelling.....	124
5.4.2	Development Of Generic Model.....	125
5.4.3	Results From Generic Model	129
5.4.5	MOST Control Channel Model Implementation	132
5.4.6	Results from Infotainment Robustness Model.....	144
5.5	Initial Conclusions on Application of Methods	160
5.5.1	Infotainment Robustness Case Initial Conclusions	160
5.5.2	Infotainment Robustness Modelling Initial Conclusions.....	161
5.6	Generic Methodology Description for New Methods.....	163
5.6.2	Generic Methodology for Robustness Modelling.....	167
Chapter 6 -	Development of New Robustness Techniques through Application to a Hybrid	
Propulsion Control System	173
6.1	Introduction	173
6.2	Hybrid Propulsion System Initialisation	173
6.2.1	Hybrid propulsion system and robustness parameter analysis.....	173
6.2.2	Hybrid propulsion System Initialisation	176

6.3	Hybrid Propulsion System Initialisation Robustness Case Development	178
6.3.1	Top Level Goal and Decomposition Strategy	179
6.3.2	System Design Robustness Argument	181
6.3.3	System Verification Robustness Argument	185
6.4	Hybrid Propulsion System Initialisation Robustness Model Development and Testing.....	187
6.4.1	Hybrid Propulsion System Initialisation Robustness Model Development	187
6.4.2	Hybrid Propulsion System Initialisation Robustness Model Testing	191
6.5	Discussion	200
6.6	Updates to Generic Methods	203
Chapter 7 – Discussion		209
7.1	Introduction	209
7.2	Targeting of Methods	209
7.3	Effectiveness	211
7.3.1	Effectiveness of Robustness Cases	211
7.3.2	Effectiveness of Robustness Modelling.....	214
7.4	Effectiveness of Overall Design for Robustness Framework	216
7.5	Cost vs. Benefit	216
7.6	Deployment of Methods.....	221
7.7	Contributions to Knowledge	224
7.7.1	Domain Knowledge of the Prevalence and Causes of Robustness Related Failures in the Area of Automotive Electronics	225
7.7.2	Design For Robustness Framework for Complex Automotive Electronic Systems	225
7.7.3	Robustness Cases	226
7.7.4	Robustness Modelling.....	227
Chapter 8 - Conclusions.....		228
8.1	Summary of Chapters.....	228
8.1.1	Summary of Chapter 1 - Introduction	228
8.1.2	Summary of Chapter 2 – Literature Review	229

8.1.3	Summary of Chapter 3 - Case Study Review of Automotive Electronics Robustness Issues	232
8.1.4	Summary of Chapter 4 - Development of Design for Robustness Framework	234
8.1.5	Summary of Chapter 5 - Development of New Robustness Techniques through Application to an Infotainment System	237
8.1.6	Summary of Chapter 6 - Evaluation of New Robustness Techniques through Application to a Hybrid Propulsion Control System	239
8.1.7	Summary of Chapter 7 – Discussion	240
8.3	Future Work	244
8.3.1	Domain knowledge capture, codification and dissemination of robustness issues 244	
8.3.3	Applicability of methods to other domains.....	245
8.3.4	Integration of Robustness Models with Other Modelling.....	245
8.3.5	Application of formal model checkers to robustness models.....	245
8.3.6	Application of Robustness Modelling Techniques to Functional Safety Analysis 246	
8.3.7	Abuse case elicitation techniques	246
REFERENCES.....		247

List of Figures

Figure 2.1 Taxonomy of dependability and security related terms from Avižienis et al (2004)	8
Figure 2.2 Error propagation from Avižienis et al (2004)	10
Figure 2.3 Comparison of robustness & conventional failures	13
Figure 2.5 Comparative analysis of source of vehicle faults from Warranty Direct Reliability Index 2013	20
Figure 2.10 EAST ADL abstraction levels and extensions (East ADL association website)	45
Figure 2.11. EAST-ADL Event Chain with associated timing constraint (ATESST2 2010)	47
Figure 2.12. EAST-ADL error model (ATESST2 2010)	48
Figure 2.13 Overview of the COMPASS toolset (Fitzgerald., Larsen, and Woodcock,(2014)	49
Figure 2.14 DEST ECS co-simulation approach to modelling embedded systems (Broenink et al, 2010)	51
Figure 2.15 FMI Concept for Model Exchange and Co-Simulation (Blochwitz et al 2012)	52
Figure 2.16 Classification Tree (Grochtmann, Grimm. and Wegener, 1993)	54
Figure 2.17 Use of Classification Trees to define TPT test cases (Bringmann and Krämer, 2006)	55
Figure 2.18 TPT test process (Bringmann and Krämer, 2006)	56
Figure 2.19 NFST approach (Becci et al, 2013)	58
Figure 2.20 Techniques the survey of formal methods application by Woodcock et al. 2009	60
Figure 3.1 Robustness failure pathology used within case study analysis	72
Figure 3.2 Elementary fault classes defined by Avižienis et al (2004)	74
Figure 3.3. Systems V-Model in automotive application	77
Figure 3.4 Primary phenomenological causes of robustness issues	82
Figure 3.5 Precipitating phenomenological causes of robustness issues	83
Figure 3.6. Electronics robustness issues - hardware and software causal areas and modified areas	86
Figure 4.1. Framework for robust design of complex automotive electronics systems	100
Figure 4.2. Design for Robustness Framework elements vs. requirements	109
Figure 5.1 MOST topology and frame structure	112

Figure 5.2 Example of Goal Structuring Notation (based on Barker, Kendall and Darlinson 1997)	115
Figure 5.3. Infotainment system robustness case top level goal	120
Figure 5.4 Infotainment system robustness case top level argument structure.....	122
Figure 5.5 - GSN robustness argument module for the Network Interface Controller.....	123
Figure 5.6 Top level structure of generic model.....	126
Figure 5.7: ECU Block from Generic Model.....	127
Figure 5.8: State Machine within ECU Block	128
Figure 5.9: Part of the state machine for distributed function availability from generic model	128
Figure 5.10: Simulation of low voltage cranking on generic model.....	129
Figure 5.11- Top level view of MOST initialisation model	133
Figure 5.12- Enlarged view of node.....	134
Figure 5.13 - Overall structure of a node in the model.....	135
Figure 5.14 Application layer initialisation states in master node.....	136
Figure 5.15 Slave node application layer behaviour during check system configuration	138
Figure 5.16 Slave node application layer behaviour during notification phase of initialisation	139
Figure 5.17 Message handler layer functions	140
Figure 5.18 NetServices layer functions	141
Figure 5.19 Network interface controller block in nodes	142
Figure 5.20 MOST Control Channel arbitration block.....	143
Figure 5.21 Comparison of message traces obtained from the model and from the physical system.....	145
Figure 5.22 Initialization status trace from MOST Control Channel model	146
Figure 5.23 Base case results	148
Figure 5.24 Test 1a Net Services for all nodes at 4ms.....	149
Figure 5.25 Test 1b Master cycle time reduced to 2ms	150
Figure 5.26 Test 1c Key nodes (IHU,HLDF,RSE) cycle time reduced to 2ms.....	151
Figure 5.27 Test 1d Key nodes (IHU,HLDF,RSE) cycle time reduced to 2ms, other nodes reduced to 3ms	152

Figure 5.28 Test 1e Key nodes (IHU,HLDF,RSE) cycle time reduced to c.2ms, other nodes c.3ms but not multiples	153
Figure 5.29 Test 2a BASE with master on 2ms Low Level retry	154
Figure 5.30 Test 2b BASE with all different retry timing - large variation.....	155
Figure 5.31 Test 2c BASE with all different retry - small variation.....	156
Figure 5.32 Test 3a Base but with low level retry increase to 15 for all nodes	157
Figure 5.33 Test 3b Differentially increase nodes with 2 HL retries in Base (IHU,HLDF,AUD,AUU,SDARS) to 15	158
Figure 5.34 Test 4 HLDF moved from 2nd to last in ring order	159
Figure 5.35 Overall test results from 9 node MOST system.....	160
Figure 5.36 Process steps in developing Modular Robustness Case	164
Figure 5.37 Generic structure of Robustness Model.....	168
Figure 5.38 Generic ECU block inputs and outputs	170
Figure 5.39 Generic ECU block internal structure	171
Figure 6.1 Parallel Hybrid Electric Vehicle driveline architecture	174
Figure 6.2 Parameter Diagram (P-Diagram) for hybrid system.....	175
Figure 6.3 Simplified network architecture diagram of HEV System.....	177
Figure 6.4 Top level goal of GSN robustness case for hybrid system initialisation	179
Figure 6.5. Decomposition strategy for top level goal of GSN robustness case for hybrid system initialisation	180
Figure 6.6 Decomposition of system design goal	182
Figure 6.7 Robustness argument module for state hang-up system design goal.....	184
Figure 6.8 System initialisation verification robustness argument.....	186
Figure 6.9. Hybrid propulsion system initialisation model - top level.....	188
Figure 6.10 Hybrid propulsion system initialisation model - ECU state machine	189
Figure 6.11 Hybrid propulsion system initialisation model – part of engine plant model state machine	190
Figure 6.12 GSN robustness argument for model based testing.....	192
Figure 6.13 Reachability test results	193
Figure 6.14 Parameter variation test result	194
Figure 6.15 3 factor parameter variation test result	195
Figure 6.16 Psuedo random start switch test	196

Figure 6.17 Fault injection test result	197
Figure 6.18 Initial correlation test result.....	198
Figure 6.19 Updated correlation test result.....	199
Figure 6.20 Modified structure of generic robustness model	204
Figure 6.21 Interaction between robustness case and robustness model	208
Figure 7.1 Resolution activities and relative levels of effort across lifecycle steps	218
Figure 7.2 Cost of implementing robust design methods vs. potential cost per issue	221
Figure 7.3 Model Based System Engineering from Forder, 2012.....	223
Figure 7.4 Simplified framework for design for robustness	223
Figure 8.1. Framework for robust design of complex automotive electronics systems.....	235
Figure 8.2 Simplified framework for design for robustness	243

List of Tables

Table 3.1 Primary impact of robustness faults in case studies on dependability attributes	78
Table 3.2 Categorization of case study faults according to the elementary fault classes	80
Table 3.3 Number of factors in activation pattern of robustness issues in case studies.....	84
Table 3.4 Specific factors in activation pattern of robustness issues in case studies	85
Table 3.5 Resulting actions to robustness issues	87
Table 3.6. Estimated earliest detection points of robustness related failures studied	89
Table 5.1 - Decomposition of the infotainment robustness case into argument modules	120
Table 5.2 MOST modelling message abstraction table	137
Table 5.3 Base Case parameter values	147
Table 5.4 Test 2b variations in low level retry timing.....	155
Table 5.5 Test 2c variations in low level retry timing.....	156
Table 6.1 Comparison of model size between infotainment & hybrid robustness models ...	201

Acknowledgements

I would like to firstly thank my Supervisor Dr. Peter Jones for his wise council and support over many years which have been essential to me both starting and finishing this thesis.

I am also grateful to Alan Curtis from WMG for his coaching and support in helping me to figure out how to combine my PhD with other work commitments.

Thanks also go to Dr. Mike Richardson, JLR Chief Technical Specialist, and Gunny Dhadyalla from WMG Embedded Systems Group for their proof-reading and feedback. I am also grateful for the helpful advice provided by Professor John McDermid, Professor Jim Woodcock and Professor Tim Kelly of the University of York.

Thanks to my friends Peter Martyres, Colin Greaves, Nick Spencer, Martin Hosmer and Steve and Linda Radmall for their (mostly) moral support and providing a distraction when I needed it.

My gratitude also goes to my mother for her support for this and all my other endeavours.

Finally, thanks to my wife Caroline for putting up with me having yet another excuse for not doing things around the house.

R. McMurran

Sept 2014

Declarations

I hereby declare that the material in this thesis has not been submitted for a higher degree at any other university. This thesis entirely contains research work carried out by R.McMurran under the supervision of Dr. R. Peter Jones, unless references are given.

Some parts of this thesis were included in the following published papers:

McMurran, R., Jones, R.P. (2010) “*Robustness Modelling of Automotive Electrical Systems of Systems*”, FISITA 2010, 30th May - 4th June 2010, Budapest, Hungary.

McMurran, R., Leslie, J. (2010) “*MOST Control Channel Modelling*”, MOST Forum 2010, 23rd March 2010, Frankfurt Germany, Reprinted in *Elektronik automotive* March 2010 Special issue MOST.

McMurran, R., Jones, R. (2013) “*Robustness Modelling of Complex Systems - Application to the Initialisation of a Hybrid Electric Vehicle Propulsion System*” SAE Technical Paper 2013-01-1231, 2013, doi:10.4271/2013-01-1231.

R. McMurran

Sept 2014

Abstract

The continual expansion in requirements for vehicle features results in a rapidly increasing complexity of automotive electronic systems. Automotive electronics exhibit properties of systems of systems including that of emergent behaviour and validation complexity. This brings with it major financial risks for automotive manufacturers due to field failures, launch delays, recalls and loss of customers. The contention of this thesis is that robustness, i.e. the ability of a system to avoid service failures resulting from external faults, is a key design criterion for automotive electronics as a mass-market system of systems. Hence effective tools and techniques for the robust design of complex automotive electronic systems are required, but initial research suggests that limited published work on robustness, as opposed to safety, has been done in this field. This thesis addresses the research question of whether a viable framework of methods to substantially improve robustness in the design of complex automotive electronics systems can be developed.

A literature review is conducted of potential methods for robust design from automotive and other domains, which identifies opportunities for contributions to knowledge in the following areas. The development of domain knowledge of the prevalence and causes of robustness related failures in the area of automotive electronics. The development of a “design for robustness” framework for complex automotive electronic systems, which should leverage best practices identified during the literature review. Particular items identified to be addressed are the adaptation of safety cases to robustness cases and the development of an approach to robustness modelling based on understanding of what are important factors to model pertaining to robustness of automotive electronics.

A review is conducted of 43 well-documented field issues in the area of automotive electronic systems. It is found that these were predominantly (60%) robustness related issues, supporting the need for improved techniques. The results confirm robustness issues as complex, interactive and emergent in nature which are generally not present during normal operation but under transient conditions, in particular during initialisation and shut-down, during failures in other systems, as a result of tolerance spread and of unforeseen (ab)use cases.

A design for robustness framework approach is developed incorporating the two proposed new methods of “robustness cases” and “robustness modelling”. A “robustness case” is a structured argument for the robustness of a system analogous to a safety case. A “robustness model” is a model based approach to early robustness verification of complex systems. These new methods are developed through their application to case study of infotainment and evaluated through subsequent application to a hybrid propulsion system. The design methods and artefacts are described in detail, including as generic approaches, and the test results from their use are shown and discussed.

Finally the viability of the methods developed and their contribution to knowledge is discussed. The knowledge gained through the study of field issues of root causes of robustness issues in automotive electronics ensured the methods were well targeted. From the application of the methods to infotainment and hybrid propulsion systems a number of positive indicators of the effectiveness of the technique are observed. An analysis is conducted of whether the likely benefits would justify the incremental costs of implementing the methods. This shows that the methods became viable at the point where they can detect a single issue which would otherwise have been undetected until final testing. Deployment approaches, known limitations and areas for further work are also described.

List of Abbreviations& Terminology

List of Abbreviations

ADL	Architecture Description Language
ACK	Acknowledged message
ASIL	Automotive Safety Integrity Level
AUTOSAR	AUTomotive Open System ARchitecture
BCM	Body Control Module
CAN	Control Area Network
COTS	Commercial Off The Shelf
CML	COMPASS Modelling Language
CTE	Classification-Tree Editor
CSP	Communicating Sequential Processes
CTM	Classification Tree Method
DFR	Design For Robustness
DFSS	Design for Six Sigma
DOE	Design Of Experiment
DTC	Diagnostic Trouble Codes
DVM	Design Verification Method
ECU	Electronic Control Unit
EM	Electric Motor
EMC	Electro Magnetic Compatibility
Fblock	Function Block
FMA	Failure Mode Avoidance
FMEA	Failure Modes and Effects Analysis

FMI	Functional Mock-up Interface
FOT	Fiber Optic Transceiver
FTA	Fault Tree Analysis
GSN	Goal Structuring Notation
HAZOP	HAZard and OPerability Study
HEV	Hybrid Electric Vehicle
HIL	Hardware In Loop
HiP-HOPS	Hierarchically Performed Hazard Origin and Propagation Studies
HLDF	High Line Display Front
HMI	Human Machine Interface
ICE	Internal Combustion Engine
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
INCOSE	International Council Of System Engineers
ISO	International Standards Organization
LIN	Local Interconnect Network
MCDC	Modified Condition/Decision Coverage
MBSE	Model Based System Engineering
MOST	Media Oriented System Transport
NAK	Non-Acknowledged message
NFST	Non-Functional Sequence Testing
NIC	Network Interface Controller
OEM	Original Equipment Manufacturer
PCM	Powertrain Control Module
POF	Plastic Optical Fiber

RSE	Rear Seat Entertainment
RTE	Run Time Environment
SAE	Society of Automotive Engineers
SBST	Search-Based Software Testing
SCA	Sequence Covering Arrays
SIL	Software In Loop
SoS	System of Systems
SPIDER	Scalable Processor Independent Design for Electromagnetic Resilience
SysML	Systems Modelling Language
TPT	Time Partition Testing
TTP	Time Triggered Protocol
UML	Unified Modelling Language
VDM	Vienna Development Method
VFB	Virtual Functional Bus
VMEA	Variation Mode and Effects Analysis

List of Terminology

Acceptably Robust	Behaviour of a system in the presence of external faults correctly implements specified services to the user which may be a limited subset defined in a degraded mode. (derived from Avižienis et al, 2004)
Availability	A system's readiness to provide correct service. (Avižienis et al, 2004)
Correctness	The degree to which a system is free from [defects] in its specification, design, and implementation. (McConnell, 2004)
Degraded Mode	When the system reacts to faults by implementing a specified subset of services to the user. (Avižienis et al, 2004)
Dependability	The ability to deliver service that can justifiably be trusted. The dependability of a system is the ability to avoid service failures that are more frequent and more severe than is acceptable. The primary attributes of dependability are availability, reliability, safety, integrity and maintainability, with robustness being a secondary attribute related specifically to external faults. (Avižienis et al, 2004)
Elusive Faults	Fault which are difficult to reproduce, also known as soft faults. (Avižienis et al, 2004)
Error	The part of the total state of the system that may lead to its subsequent service failure. (Avižienis et al, 2004)
Error Detection	To detect a deviation from the correct system state. (Avižienis et al, 2004)
Failure	An event that occurs when the delivered service deviates from correct service - abbreviation of service failure. (Avižienis et al, 2004)

Fault	The cause of an error is a fault, which can be internal or external to the system. Internal faults originate outside the system boundary, external faults originate outside the system boundary and propagate errors into the system by interaction or interference. (Avižienis et al, 2004)
Faults Prevention	To prevent the occurrence or introduction of faults. (Avižienis et al, 2004)
Fault Tolerance	To avoid service failures in the presence of faults. (Avižienis et al, 2004)
Fault Removal	To reduce the number and severity of faults. (Avižienis et al, 2004)
Fault Forecasting	To estimate the present number, the future incidence, and the likely consequences of faults. (Avižienis et al, 2004)
Hard Faults	Fault which are easily reproduced, also known as solid faults. (Avižienis et al, 2004)
Integrity	The absence of improper system alterations. (Avižienis et al, 2004)
Maintainability	The ability to undergo modifications, and repairs. (Avižienis et al, 2004)
Partial Failure	When the system may react to faults by adopting a degraded mode implementing a specified subset of services to the user. (Avižienis et al, 2004)
Powermode	Enumerated status parameter indicating the virtual key position on vehicles with keyless ignition , e.g. 0 indicates shut down (key out), 6 ignition on with engine off, 9 engine cranking.
Reliability	The ability to provide continuity of correct service. (Avižienis et al, 2004)
Reproducibility	at the ability to identify the activation pattern of a fault that had caused one or more errors. (Avižienis et al, 2004)
Robustness	Dependability with respect to external faults. (Avižienis et al, 2004)

Robustness Fault	An internal fault within a system that is instigated by an external fault or activation conditions. (derived from Avižienis et al, 2004)
Robustness-related Failure	Service failure as a result of a fault external to the service delivery system. (derived from Avižienis et al, 2004)
Safety	The absence of catastrophic consequences on the user(s) and the environment. (Avižienis et al, 2004)
Service Failure	An event that occurs when the delivered service deviates from correct service - often abbreviated to failure. (Avižienis et al, 2004)
Quiescent Current	Drain on the vehicle battery from electronic control units in their inactive or dormant state, typically entered when ignition is switched off.
Verification	The process of checking whether the system adheres to given properties. (Avižienis et al, 2004)
Vulnerability	An internal fault that is dormant until an external fault occurs makes it active. (Avižienis et al, 2004)

Chapter 1 – Introduction

1.1 Background

The complexity of electronic systems on premium vehicles continues to increase with over 50 Electronic Control units on a premium vehicle and typically 100M lines of code and is projected further increase to support the features required for the next generation of vehicles (Charette, 2009, Mössinger, 2010). This is creating major financial costs due to field failures and even larger risks for automotive companies through launch delay, product recalls, warranty costs and loss of customers. The scale of the risk is huge running into tens and hundreds of millions of pounds. While safety-related recalls, such as Toyota's 2010 recall of 430,000 vehicles related to the software that controls hybrid braking systems (Toyota, 2011 p58), receive high levels of publicity, the robustness of electrical and electronic systems is concern for all systems. J.D. Power in their 2013 Vehicle Dependability Survey (J.D.Power, 2013) found that brand loyalty among owners of premium models declines from 55% to 39% if they have experienced three or more problems with their vehicle. Such a loss of repeat customers has serious financial implications for vehicle manufacturers.

The availability of tools and techniques for the design of robust complex automotive electronic systems will significantly allay the financial risk to automotive companies, whilst giving them a major competitive advantage through increasing their capacity to deliver sophisticated and innovative products.

In recent years the phenomenon of Systems of Systems has become recognised as a specific type of Complex System affecting many fields including biology, sociology, engineering and military where new systems are composed of a number of individual systems

with their own levels of autonomy (Bar-Yam et al 2004). There are a number definitions of system of systems (Maier 1998), (Boardman and Sauser 1996), (Bar-Yam et al 2004) but significant properties are autonomy of elements, evolutionary development, heterogeneous elements, sharing of resources, geographic distribution and emergent behaviour.

Emergent behaviour is an unanticipated interaction between systems which can be either positive or negative. The potential for positive behaviour has been utilised in software systems based on collaborating agents to outperform conventional distributed control systems (Li, Sim & Low 2006). Negative behaviour can result in partial or full loss of service by one or more systems within the System or Systems.

Automotive electrical and electronics systems can also be described in terms of a System of Systems (McMurrin 2006) which exhibits these properties but also has the resultant issues of emergent behaviour and validation complexity as the state space of the overall system increases exponentially. The contention of this thesis is that robustness, i.e. the ability of a system to avoid service failures resulting from external faults, is a key design criteria for automotive electronics as a mass-market system of systems.

Techniques for achieving robust electrical and software based systems within automotive are a major issue for the industry as the traditional techniques, based around component or individual systems, are proving increasingly limited as system complexity increases

Initial review of published work suggests that little work has been done within the automotive electronics field on designing for robustness at a Systems of Systems level. There are some approaches being taken within other fields and there are some existing practices which may be extensible but no clear framework has been established for this. The contribution of this work is the establishment and proving of such a framework and the development of new techniques supporting this approach.

1.2 Objectives

The high level research question this thesis addresses is whether a framework of methods to substantially improve robustness in the design of complex automotive electronics systems can be developed which is viable, in terms of cost and time versus expected benefits. This should result in both fewer failure modes in new designs and products which are resilient to external faults. This framework will integrate state of the art methods in automotive development and develop and adapt methods from other domains.

To achieve this aim of developing a framework for robust design of complex automotive electronics a number of objectives must be achieved. Firstly the actual root causes of issues due to lack of robustness in design must be understood to ensure the focus of the work is correct. This will establish a context to understand the strengths and weaknesses of current techniques and tools used in automotive development and to review the applicability of tools and techniques from other domains. This then should enable the research and proposal of a framework for design for robustness and the identification of requirements for the development of new methods to support this framework. A major part of this work is to develop new approaches for the automotive domain which have the most potential to enhance capability for robust design. Two specific new methods are developed within this thesis. Firstly “*robustness cases*” which are structured arguments for the robustness of a system or system of systems analogous to a safety case. Secondly a method of “*robustness modelling*” to allow analysis of properties relevant to robustness of a system or system of systems is developed. Finally the added value and viability of the new methods and framework should be assessed and quantified.

1.3 Structure

This thesis is structured as follows:

Chapter 2 reviews and appraises background literature on automotive electronics, complex systems, techniques used in their design, including modelling and safety cases, and validation methods covering the state of the art approaches within automotive but also in other relevant fields particularly aerospace and defense.

Chapter 3 gives the results of a review of 43 case studies undertaken to understand the nature of issues in automotive electronic systems which have not been detected by normal design and development methods to confirm if robustness is indeed a significant factor and issues with current approaches.

Chapter 4 utilises understanding of the actual causes of robustness issues in complex automotive electronics and the knowledge of applicable techniques derived from the literature review to propose a Design for Robustness Framework for automotive electronics.

Chapter 5 develops the two new methods proposed in the Design for Robustness Framework of “*robustness modelling*” and “*robustness cases*” through their application to the design of the control channel of an infotainment system networked by a MOST high speed optical-fiber ring.

Chapter 6 evaluates and demonstrates the wider applicability of the two methods developed in Chapter 5 through their application in combination to the initialisation of a hybrid electric vehicle control system.

Chapter 7 discusses the findings from the work, specifically whether it answers the research question of a viable framework of methods to substantially improve robustness in the

design of complex automotive electronics systems can be developed and considers where specifically contributions to knowledge have been achieved.

Chapter 8 summarises key conclusions from the thesis and highlights areas of future work to improve the robustness of complex systems.

Chapter 2 – Literature Review

2.1 Introduction

In this chapter a literature review is conducted to: define terminology, understand the nature of complex systems and review potential methods which could be employed to improve their robustness. Initially the review seeks to clarify concepts of robustness and complex systems of systems. Subsequently there is a review and appraisal of background literature on automotive electronics to confirm the level of complexity, associated issues and existing approaches to managing complexity. Next the state of the art approaches for the design and validation of complex systems are reviewed, including modelling and safety cases, not only from automotive but also in other relevant fields particularly aerospace and defense. Finally conclusions are drawn regarding where the most significant opportunities are for new research within the context of this thesis.

2.2 Robustness

As this thesis is fundamentally about robustness it is important to understand what is meant by the concept of robustness in this context and in particular what distinguishes this from similar concepts such as reliability and dependability.

Avižienis, Laprie, Randell and Landwehr (2004) propose definitions of dependability and related concepts as well as a taxonomy showing how they are related. Their stated intent in doing this was to give a set of definitions that would enable communications across scientific and technical communities in the field of dependable and secure computing and these definitions are now widely cited. They class robustness as a specialized secondary attribute of dependability “*dependability with respect to external faults, which characterizes*

a system reaction to a specific class of faults”, although they do not elaborate further on the specific class of faults. From a software perspective McConnell (2004) defines the concepts of correctness, robustness and reliability as follows.

“Correctness: The degree to which a system is free from [defects] in its specification, design, and implementation. Robustness: The degree to which a system continues to function in the presence of invalid inputs or stressful environmental conditions. Reliability: The ability of a system to perform its requested functions under stated conditions whenever required - having a long mean time between failures.”

Similar definitions are also given by Meyer (1997) for correctness relating to meeting specifications and robustness being the ability to react appropriately to abnormal conditions, but he defines reliability as being *a concern encompassing correctness and robustness*”..

The standard for automotive functional safety ISO26262 (ISO, 2011) defines a robust design as *“a design that has the ability to function correctly in the presence of invalid inputs or stressful environmental conditions”*.

Coming from a mechanical perspective Wang, Wu, and Lust (1997) state *“robustness can be defined in terms of reliability being least sensitive to variations in the statistics (such as means and standard deviations) of the random variables while still meeting design cost and performance targets”*.

Common to both mechanical and software definitions is that robustness is more concerned with ensuring specified functionality is maintained over the widest possible input conditions, including variation in mechanical parts and fault conditions, as opposed to reliability which is concerned with reducing the occurrence of faults in the first case. However it is also clear that robustness must be defined relative to other concepts such as dependability, reliability, faults and failures to be able to use an unambiguous fashion. Avižienis et al (2004) in defining an overall taxonomy of terms related to dependability and

security provides a good basis for doing this. This taxonomy splits into three key areas of: the attributes of dependability and security, the threats to dependability and security and the means of achieving dependability and security. This is illustrated in Figure 2.1 and will be subsequently explained adopting the same approach as the authors in highlighting key terms in bold.

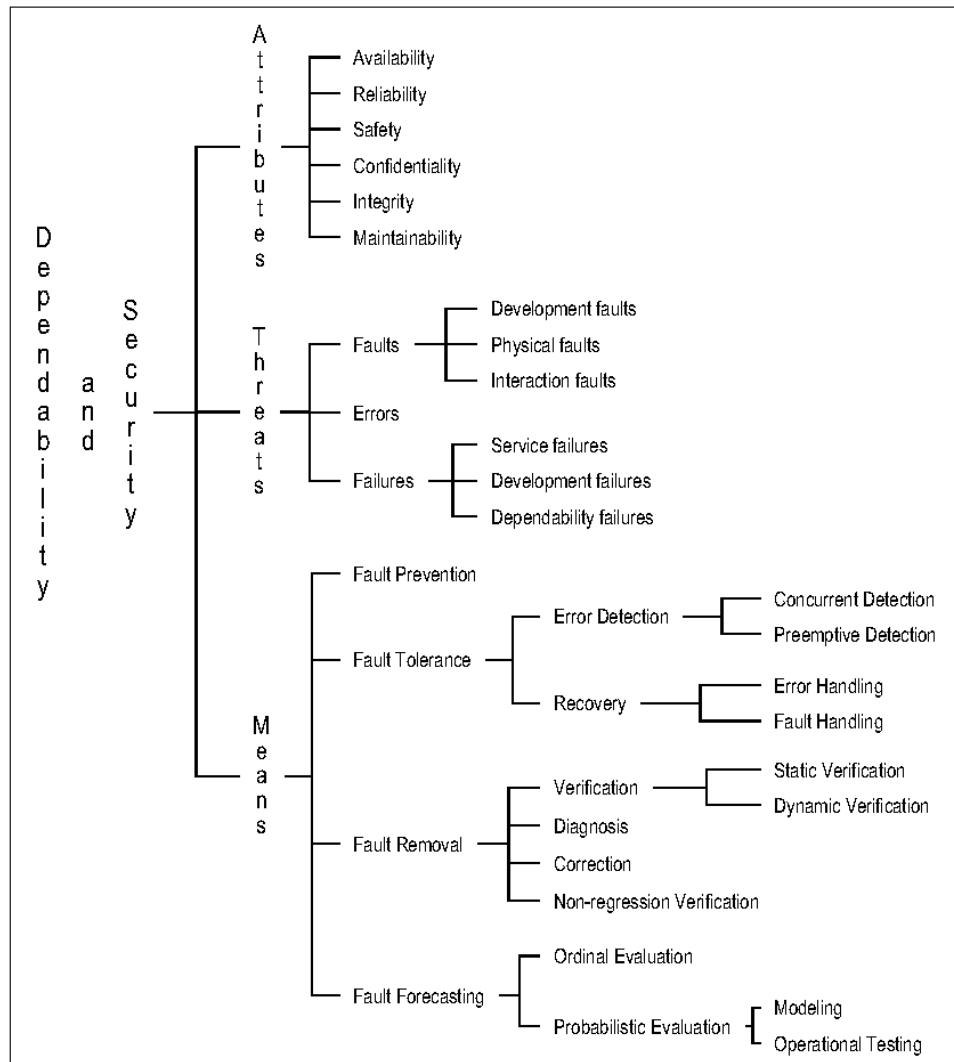


Figure 2.1 Taxonomy of dependability and security related terms from Avižienis et al (2004)

Avižienis et al (2004) give two definitions of dependability. They give the original definition of **dependability** as “*the ability to deliver service that can justifiably be trusted*”. However they then go on to give an alternate definition as “*the dependability of a system is*

the ability to avoid service failures that are more frequent and more severe than is acceptable". These definitions are compatible but with differing emphasis. The first definition focusses on justification of trust, while the second on the criterion for what is acceptable, clearly both of these aspects need to be considered.

They then list what they consider to be the primary attributes related to dependability and security which they define as:

- *"**availability**: readiness for correct service;"*
- *"**reliability**: continuity of correct service;"*
- *"**safety**: absence of catastrophic consequences on the user(s) and the environment;"*
- *"**integrity**: absence of improper system alterations;"*
- *"**maintainability**: ability to undergo modifications, and repairs."*

They then examine the threats to dependability and security in the relationships between them. A **service failure**, often abbreviated to **failure**, is defined as *'an event that occurs when the delivered service deviates from correct service'*". They note that this deviation can be due to a system not complying with its functional specification or because the functional specification did not specify required system functions adequately to implement the desired service. They also note that service failures can vary in time (**service outage**), may assume different forms (**service failure modes**) and have varying levels of consequence (**failure severities**). They also highlight **partial failures** whereby the system may react to faults by adopting a **degraded mode** implementing a specified subset of services to the user. Their classification scheme does not restrict failures to service failures but also includes **development failures** whereby a system does not reach operation as the development is never completed and **dependability failures** whereby service failures occur at an unacceptable level.

They define an **error** as “*the part of the total state of the system that may lead to its subsequent service failure*”. The cause of an error is a **fault**, which can be internal or external to the system. They classify faults under 3 overlapping groups of **development faults** which occur during the development of a system, **physical faults** that affect hardware and **interaction faults** that include all external faults. They introduce a more detailed classification of faults which will be used in Chapter 3 to classify faults seen on real systems. They also note that internal faults can be dormant until an external fault occurs makes it active which they term a **vulnerability**.

This then creates a relationship (or “*pathology of failure*”) between faults, errors and failures, illustrated in Figure 2.2, whereby an internal or external fault can lead to an error which can lead to service failure either directly or indirectly through causing other errors. This service failure can itself propagate to other system components which are dependent on the service causing input errors that can lead to service failures.

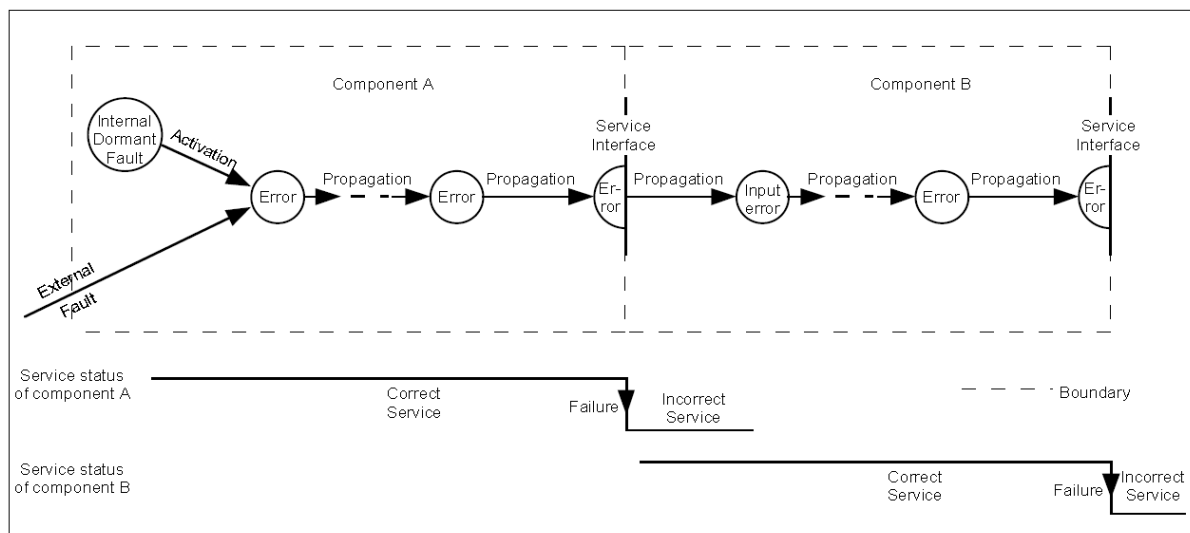


Figure 2.2 Error propagation from Avižienis et al (2004)

Avižienis et al then introduce the concept of **reproducibility** as “*the ability to identify the activation pattern of a fault that had caused one or more errors*”, then differentiate between faults which are easily reproduced as **solid** or **hard** faults and those which are difficult to reproduce as **elusive** or **soft** faults. They go on to note that “*most residual development faults in large and complex software are elusive faults: they are intricate enough that their activation conditions depend on complex combinations of internal state and external requests, that occur rarely and can be very difficult to reproduce*”.

Last area of the taxonomy deals with the means to achieve dependability and security which are split into 4 categories:

- “**fault prevention**: means to prevent the occurrence or introduction of faults;”
- “**fault tolerance**: means to avoid service failures in the presence of faults;”
- “**fault removal**: means to reduce the number and severity of faults;”
- “**fault forecasting**: means to estimate the present number, the future incidence, and the likely consequences of faults.”

Fault tolerance requires **error detection** to detect a deviation from the correct system state which can be done during normal service delivery (**concurrent detection**) checking for errors caused by active faults or while normal service delivery suspended (**pre-emptive detection**) checking for errors caused by transient errors or dormant faults. Having detected an error then a **recovery** mechanism is required to return the system to a correct state. This can be achieved by **error handling** to move the system to a correct state and **fault handling** to prevent faults re-occurring.

Terms are identified for four activities relating to fault removal. **Verification** is defined as “*the process of checking whether the system adheres to given properties*” and has sub-classes of **static verification** and **dynamic verification** depending on whether the actual system is exercised. Under this definition model based testing is a type of static verification.

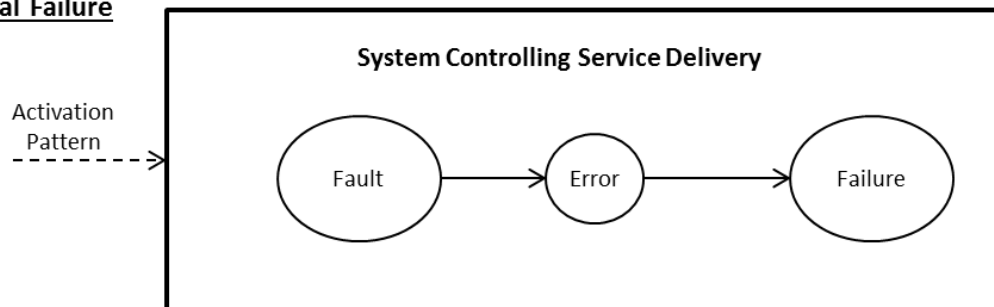
If required properties are not adhered to then **diagnosis** and **correction** has to occur to identify and remove the underlying fault or mechanism that allows fault to lead to an error. Having corrected a fault it is necessary to perform **non-regression verification** to ensure there are no side-effects of the change. **Fault forecasting** is defined as “*performing an evaluation of the system behavior with respect to fault*”. This can be done using an **ordinal or qualitative evaluation** that aims to identify, classify and rank mechanisms that would lead to failures or through **probabilistic evaluation** which evaluates the probability of attributes being satisfied through either modelling or operational testing.

ISO26262 (ISO 2011) a significant standard in the automotive domain is aligned with the definitions given by Avižienis et al. It also follows a fault, error, failure causality paradigm. A fault is defined as an “*abnormal condition that can cause an element or an item to fail*” and it is noted that faults can be permanent, intermittent or transient in nature. It defines an error as a “*discrepancy between a computed, observed or measured value or condition, and the true, specified or theoretically correct value or condition*”. Hence an error is defined as a state of the system not necessarily as a cause of failure or as a consequence to the user and it is noted that an error can be due to a fault or unforeseen operating condition, i.e. incomplete specification. A failure is defined as the “*termination of the ability of an element to perform a function as required*” and again notes that incorrect specification is a source of failure. An incorrect specification would be one which fails to request the desired behaviour of a system through misinterpretation or omission of stakeholder requirements. Given this correlation of definitions it is reasonable to use the definitions given by Avižienis et al in the domain context of this thesis.

Hence for this thesis the definition of robustness used will be that given by Avižienis et al of **robustness** being “*the dependability of a system with respect to external faults*”. Under this definition it is still equivalent to McConnell’s (2004) definition of robustness as “*the*

degree to which a system continues to function in the presence of invalid inputs or stressful environmental conditions”, stressful environmental conditions being a mechanism for a physical faults or interaction faults in the taxonomy given by Avižienis et al. Hence it can be inferred that a robust system is one that is resilient to external faults. Based on error propagation models in Avižienis et al (2004), Figure 2.3 illustrates this definition of a robustness failure being the result of an external fault to the system delivering the service. The thick black box represents the boundary of the system controlling the service delivery, the solid arrows show the error and fault propagation and the dashed arrows show activation patterns. It is important to note the distinction between a robustness failure which must have an error propagated by an external fault and a failure due to an elusive internal fault in which the fault may be difficult to reproduce, due to a complex activation pattern or a low failure rate, but does not contain an input error with respect to the specified interface.

Conventional Failure



Robustness Failure

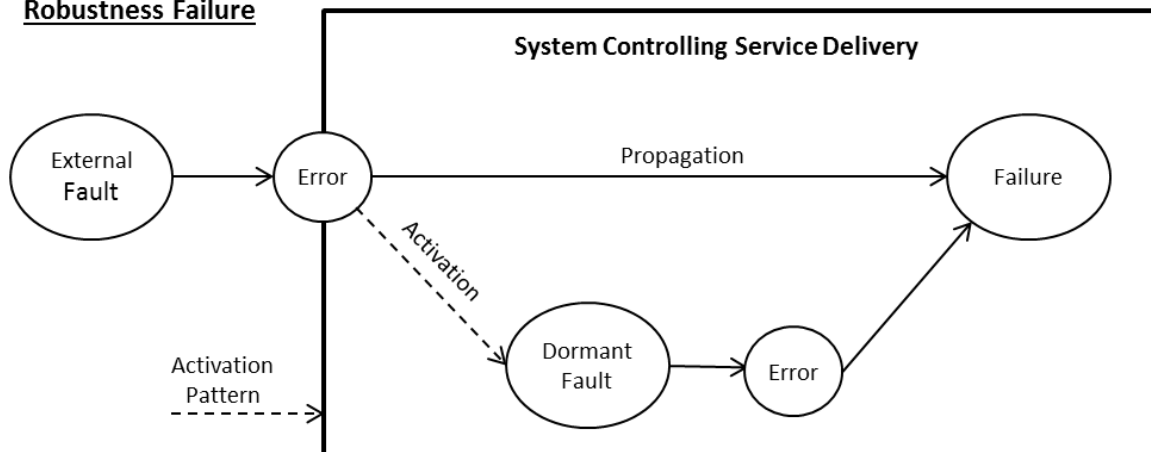


Figure 2.3 Comparison of robustness & conventional failures.

From the definitions given by Avižienis et al (2004) further definitions can be derived to be used in this thesis. A **robustness fault** is an internal fault within a system that is instigated by an external fault or activation conditions. This is noting that the failure mechanism may be through triggering a dormant internal fault. An **acceptably robust** system is one whose behaviour in the presence of external faults correctly implements specified services to the user which may be a limited subset defined in a degraded mode. A **robustness-related failure** is a service failure as a result of a fault external to the service delivery system.

2.3 Complex Systems of Systems

Most sources agree that what distinguishes the complex from the complicated is the property of interconnectedness i.e. having many connections between the parts or elements. This is most well defined by Lissack and Roos (2000) who cite the origin of the syllable “plex” in the word “complex” as deriving from the Latin to weave. This interconnectedness creates difficulty in modelling and predicting behaviours of such systems which has created a whole new discipline of Complexity Science concerned with the concept of emergence.

The origins of Complexity Science can be traced back to the 1940s notably a landmark paper “Science and Complexity” by Warren Weaver (1948). In this paper Weaver shows the difference between complicated problems, which he refers to as “*disorganised complexity*” which are amenable to statistical analysis, and complex problems, which he refers to as “*organised complexity*” which are not. Weaver goes on to state, with considerable foresight, that two key wartime advances of computing and multidisciplinary approaches may hold the key to tackling such problems.

Since Weaver's paper there has been a massive amount of, mainly theoretical, work in the field of Complexity Science. A good overview of Complexity Theory is given by Lucas (2000) in which he states that "*Complexity Theory states that critically interacting components self-organise to form potentially evolving structures exhibiting a hierarchy of emergent system properties*". He goes on to state that "*this theory takes the view that systems are best regarded as wholes, and studied at such, rejecting that additional emphasis on simplification and reduction as inadequate techniques*" due to the inherent nonlinearity of strongly interconnected systems.

Lucas goes on to describe four different types of complexity. Type 1 is "*Static Complexity*" which is concerned with invariant structures such as hardware. Type 2 is "*Dynamic Complexity*" which recognises the temporal nature of dynamic interactions. Type 3 is "*Evolving Complexity*" which recognises organic behaviour within a system. Type 4 is "*Self-Organising Complexity*" which recognises the ability of systems to co-evolve with their environment.

In recent years a specific type of complex system has been identified as Systems of Systems (SoS) which are comprised by linking of a number of existing systems together to deliver higher levels of functionality or to share resources. Such Systems of Systems are becoming increasingly prevalent in engineered systems through using local and wide area communications. Despotou (2011) in his paper "*Introducing the Challenges of Dependable Systems of Systems*" discusses the difficulty in making claims (particularly safety related) about the overall performance of a System of Systems pointing out that "*failures in a system may propagate through collaboration between elements and transform into different types of failures, eventually manifesting as hazards*".

Hence systems with planned interactions exhibit complexity that increases with the number and type of interactions, resulting in emergent behaviours including fault propagation

which are very difficult to predict. The next section on automotive electronics will review technology approaches being taken in this field, consider which of these types of complexity is present in current and future systems and the degree to which this is affecting the automotive industry.

2.4 Automotive Electronic Systems

2.4.1 Complexity of Automotive Electronic Systems

A typical Electronic Architecture for a premium vehicle is illustrated in Figure 2.4 based on examples from Audi (Marstaller 2013), BMW (Reichart and Heinecke 2006) and Jaguar-Land-Rover (Rivett 2013). The Electronic Control Units (ECUs) are typically networked by several domain-orientated CAN (Controller Area Network) buses connected via a Gateway module, a high-speed infotainment bus such as MOST (Media Orientated System Transport) and a number of sub buses typically using the LIN (Local Interconnect Network) protocol.

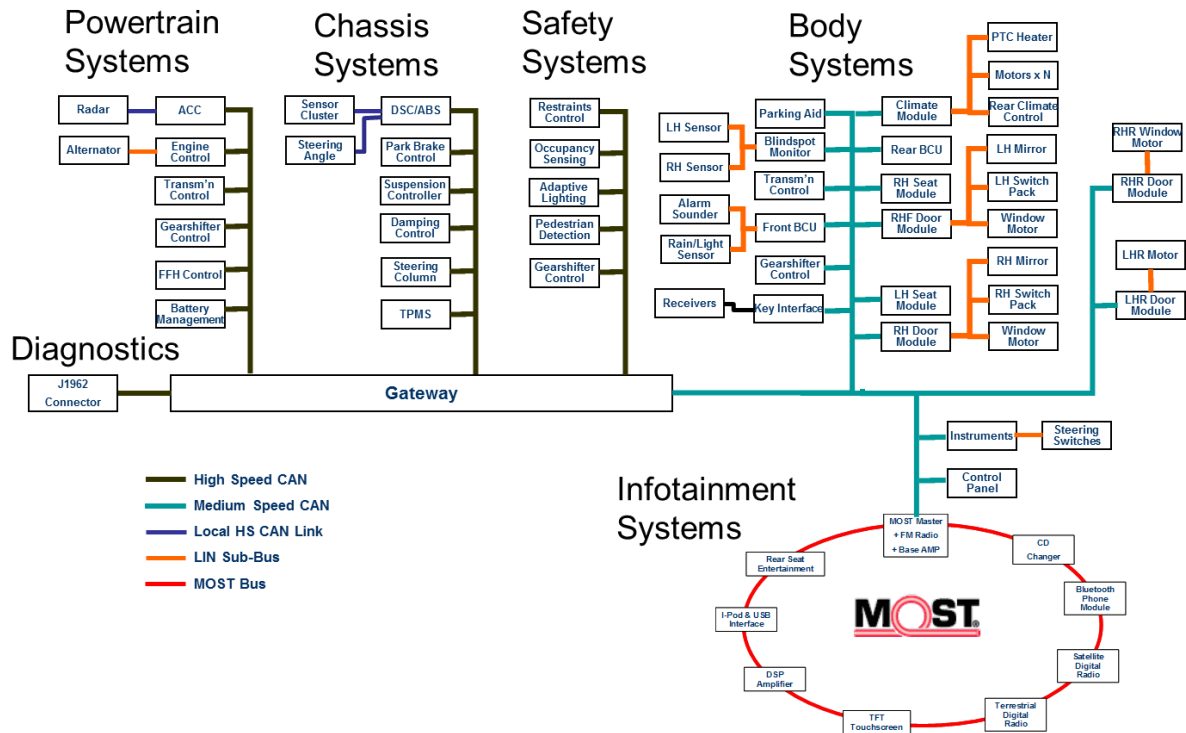


Figure 2.4 – Typical electronic architecture for a premium vehicle

Estimates suggest that a modern premium vehicle has between 70 and 100 ECUs and around 100M lines of code (Charette, 2009, Mössinger, 2010). An earlier figure for premium cars is up to 70 ECUs in 2004 but noting a trend for a tenfold increase of memory usage every 4 years on BMW products (Frischkorn 2004). This highlights that the major increase of automotive complexity is in software based functions rather than hardware where the overall number of ECU is no longer increasing significantly but each has increasing levels of functionality. This is supported by the increasing processing power and memory size of ECUs which are becoming open platforms for multiple software applications rather than dedicated to a particular function and through increasing networking bandwidth between ECUs (Mössinger, 2010).

Automotive electronics has been shown to exhibit the commonly agreed properties of a system of systems including autonomy of elements, evolutionary development,

heterogeneous elements, sharing of resources, geographic distribution and emergent behaviour (McMurrin, McKinney, Tudor and Milam 2006).

A further dimension of the complexity of automotive electronic systems is through being real-time embedded systems. Herman Kopetz, one of the foremost authorities on the design of distributed systems, defines a real-time computer as “*one where the correctness of the system behaviour depends not only on the logical results of the computations but also on the time when these are produced*” (Kopetz 2011, p19). He goes on to note that real-time computer systems are always part of larger real-time systems and that these are often part of distributed systems where nodes are connected by a real-time communication network. He also makes the distinction between hard real-time systems where deadlines must be met under all conditions and soft ones where there are no deadlines or it is permissible to miss deadlines occasionally. He also describes embedded systems as products with real-time computer based control systems and notes typical characteristics of these including; being designed for a mass market, having a man-machine interface, a tendency towards static structures analyzable at design time that are more robust, a maintenance strategy and limited amounts of energy. From these definitions it can be seen that an automotive electronics network is a distributed, hard real-time, embedded system as it encompasses hard deadlines in interacting with users and systems under control such as engines, brakes, etc. and makes extensive use of real-time communications such as CAN busses.

2.4.2 Impact of Increasing Complexity of Automotive Electronic Systems

The scale and seriousness of the issues of managing complex software faced by the automotive industry is underlined by the following comments made by Mark Chernoby, the Senior VP for Engineering of Chrysler, in a recent interview reported by Ponticel (2013).

“By far the biggest challenge right now is managing software. The software on vehicle has grown to millions and millions of lines of code, done by a wide range of suppliers, some of it internal, some of it external, and each one off in a corner doing their own little work. Getting this software to work right, to talk together at the right time, to perform the right functions, by far I think is the biggest challenge we face today, in my view.”

The problems of increasing complexity are also recognised by key industry and academic leaders in Europe too including Heinecke et al (2008) who state *“The OEMs are generally struggling to understand and control the emerging behavior of the complex distributed functions, resulting from the integration of subsystems.”*

Kopetz (2011 p19) states *"Automotive manufacturers view the proper exploitation of computer technology as a key competitive element in the never ending quest for increased vehicle performance and reduced manufacturing costs"*. This is reflected in some manufacturers setting up specific divisions to focus on these technologies e.g. BMW Car IT GmbH (www.bmw-carit.com), Audi Electronics Venture GmbH (www.audi-electronics-venture.com) and VW subsidiary Carmeq GmbH (www.carmeq.de).

The problem for automotive is not just one of increasing complexity but also compounded by the relatively short lifecycles for such a complex product. Heinecke (2010) from BMW refers to this as “Dynaxity” a portmanteau of dynamic, which he relates to rate of change and complexity, which he relates to the number of system elements and the number and type of connections. Heinecke also notes the differences in lifecycles between different functional areas of the vehicle, especially those driven by consumer electronics that will require continual updates. The updating of parts of a complex system then leads to problems of ensuring compatibility with the existing parts (Frischkorn, H., 2004, Pretschner, Broy, Krüger and Stauner, 2007).

However whilst the general issues of increasing complexity of automotive electronic systems are acknowledged, the industry is naturally reticent to publish specific figures relating to the incidence and cost of these issues. To give a quantitative view on the incidence of issues published data from Warranty Direct Reliability Index (2013) was analyzed. The data covers some 50,000 vehicles they have under warranty of varying age and mileage providing they have at least 50 examples of particular model. There is a breakdown of the average percentage of faults in vehicles under warranty into component areas which was used to do a comparative analysis shown in Figure 2.5.

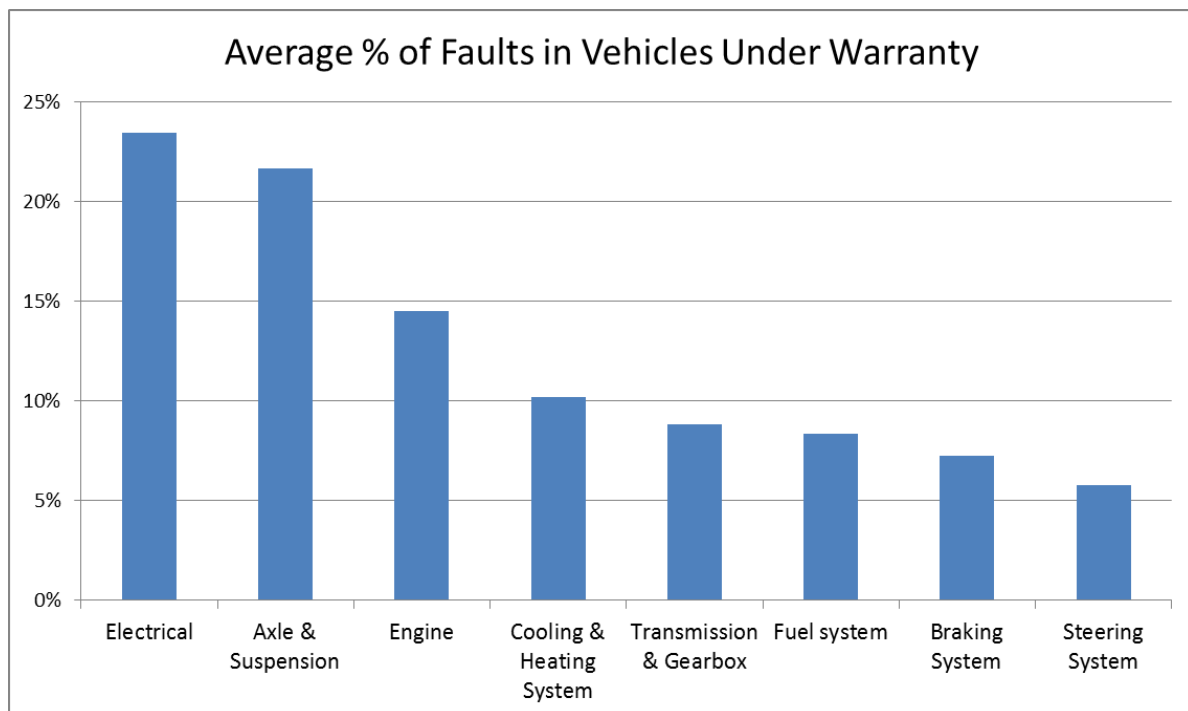


Figure 2.5 Comparative analysis of source of vehicle faults from Warranty Direct Reliability Index 2013

The results show electrical components to be the biggest source of failure by some margin noting that the high figures for axle & suspension include items which would be expected to wear out such as joints, rubber boots and shock absorbers. What these results will

not show is the early life failures which are covered under the manufacturer's warranty; it is likely software related failures would be detected and rectified during this period.

Further evidence of issues in the public domain is where manufacturers have to do safety-related recalls to proactively fix a fault. Recalls are a major issue for car manufacturers in terms of customer dissatisfaction, loss of brand image and cost. There have been several high profile recalls due to safety related software issues (Charette, R. N. 2009) most recently that GM are recalling about 33,700 vehicles to fix a software issue (Reuters 2013). As the vast majority of software in a vehicle is not safety-related and not subject to recalls, issues are not in the public domain but it is reasonable to assume that these occur at far higher incidence rates than safety related issues due to greater amount of code and not being subject to as high a degree of rigour in development process. This assessment is also shared by partners in the ATESSST project which included OEMs and Tier 1 suppliers who state *"as much as 35% of vehicle failures today stem from errors related to embedded systems, forecasted to reach 63% in 2010 if no action plan is considered"* (Cuenot et al 2007).

2.4.3 Existing approaches to managing Complexity of Automotive Electronic Systems

Two prominent industry figures Mössinger (2010) of Robert Bosch and Heinecke (2010) of BMW both conclude that to cope with complexity and rate of change, standards for open systems and sharing and reuse of software are absolutely vital, Heinecke referring to it as "the Survival Strategy".

The major automotive initiative in this respect is AUTOSAR (Automotive Open System Architecture), which aims to create a standard operating environment of basic software as illustrated in Figure 2.6 (www.autosar.org). On top of standardized basic software components AUTOSAR middleware gives a run time environment (RTE) which is common

to all ECUs and supports the sharing, reuse and flexible deployment of software components with an AUTOSAR interface.

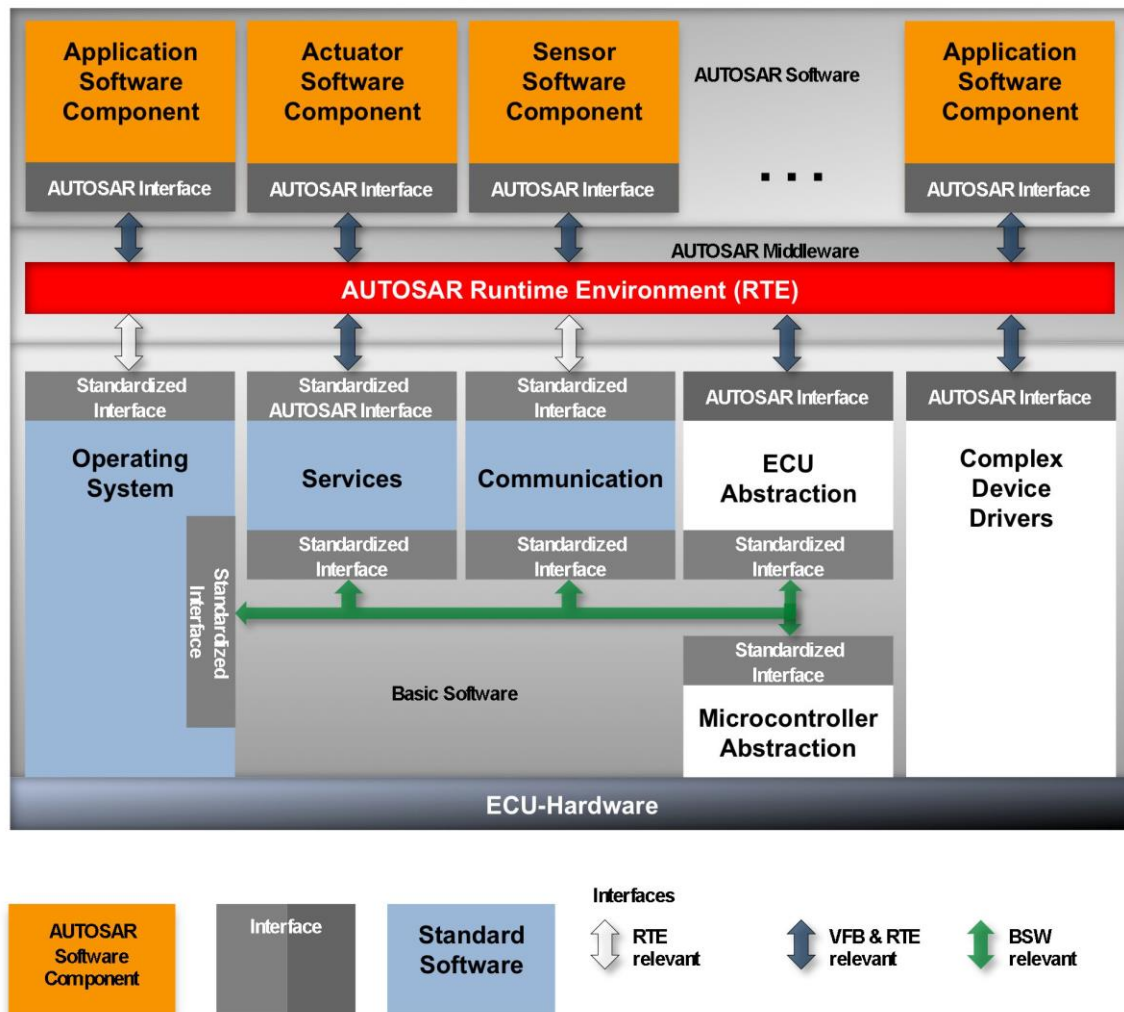


Figure 2.6 AUTOSAR Automotive Open System Architecture (www.autosar.org)

As well as the base layer software and run time environment AUTOSAR also includes an underlying methodology and toolset for the deployment of software functions. This is done initially through linking software components through a Virtual Functional Bus (VFB) which abstracts communication layer away from the application software hence improving composability (the ability to guarantee that a component property is preserved when

integrated) and reusability of software components (Heinecke, Damm, Josko, Metzner, Kopetz, Sangiovanni-Vincentelli, and Di Natale, 2008). This is illustrated in Figure 2.7.

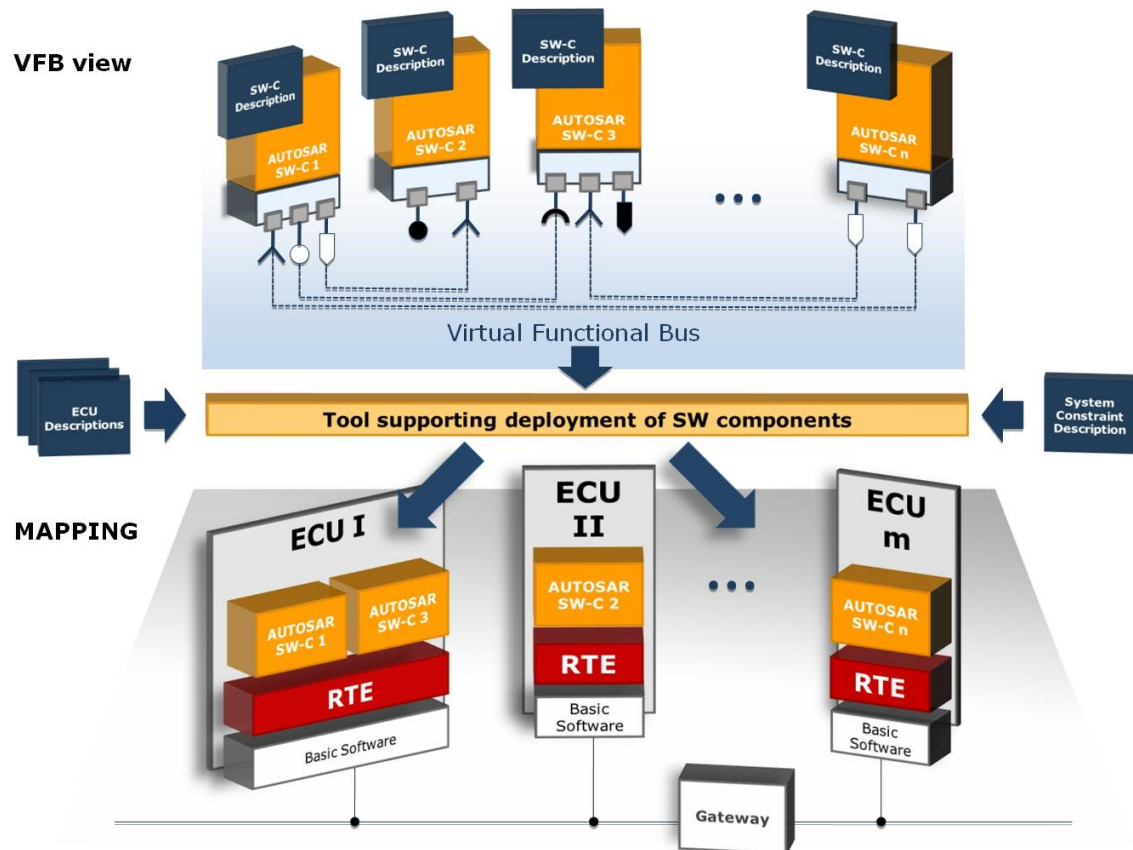


Figure 2.7 AUTOSAR process for deploying standard software components (AUTOSAR 2013)

The AUTOSAR methodology uses a series of design artefacts, such as ECU descriptions, system constraint descriptions and interface databases, and tools which enforce a contractual approach to the interactions between software functions. In a related initiative called Artop, an AUTOSAR Tool Platform or ecosystem is being developed to create an integrated development environment (IDE) for standard AUTOSAR tools and more advanced tools to enhance the design and development process (Knüchel, Rudorfer, Voget, Eberle,

Sezestre and Loyer 2010). The structure for the Artop tool platform is illustrated in Figure 2.8.

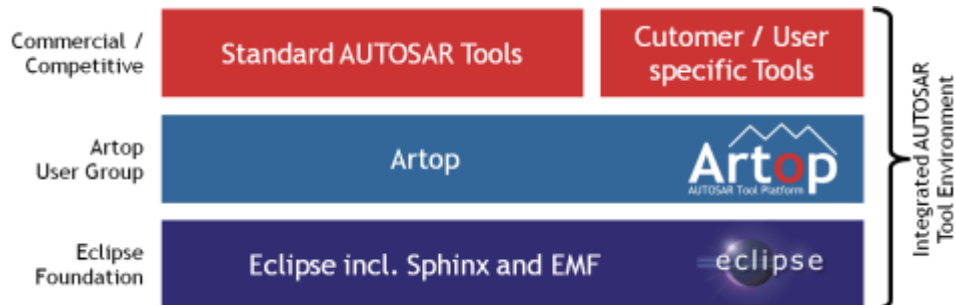


Figure 2.8 Artop AUTOSAR Tool Platform (Artop 2013)

AUTOSAR is not the only automotive open-source initiative. The GENIVI initiative is driving the broad adoption of an In-Vehicle Infotainment (IVI) open-source development platform based on a Linux operating system (GENIVI 2013). From a robustness perspective the opportunity from having standardised software elements must be balanced against the risks of taking commercial off-the-shelf software from a wider and potentially less mature supplier base.

Another area highlighted by Mössinger (2010) and Heinecke (2010) where there are significant developments is that area of higher bandwidth and more deterministic networking protocols such as Flexray and Ethernet. Heinecke et al (2008) note that these busses can support new more integrated topologies with less *physical* interconnections and ECUs and supporting the partitioning of a single physical communication channel to reduce temporal interference and improve error containment.

Flexray was first introduced by BMW, Schedl (2007) gives the motivation for this is be driven by CAN bus bandwidth limitations, unacceptable delays caused by multiple gateways

and unacceptable levels of complexity using multiple CAN buses. This led to additional costs for system integration and warranty and a need for deterministic behaviour. More recent work (Scheickl, Ainhauser, Gliwa, 2012) has used timing extensions from the AUTOSAR environment to apply tools for the specification and verification of timing constraints to optimally exploit Flexray network's deterministic capabilities.

Hyung-Taek, Volker and Herrscher (2011) note there is a gathering pace for higher bandwidth and lower cost busses initially for diagnostics and software updates and as a lower cost higher bandwidth infotainment bus but future potential to provide additional high speed data communication between larger ECUs, for instance for future driver assistance applications. Figure 2.9 illustrates a roll-out strategy for Automotive Ethernet described by Hank, Suermann and Müller (2012) culminating an automotive Time-Triggered version providing a high bandwidth deterministic network backbone.

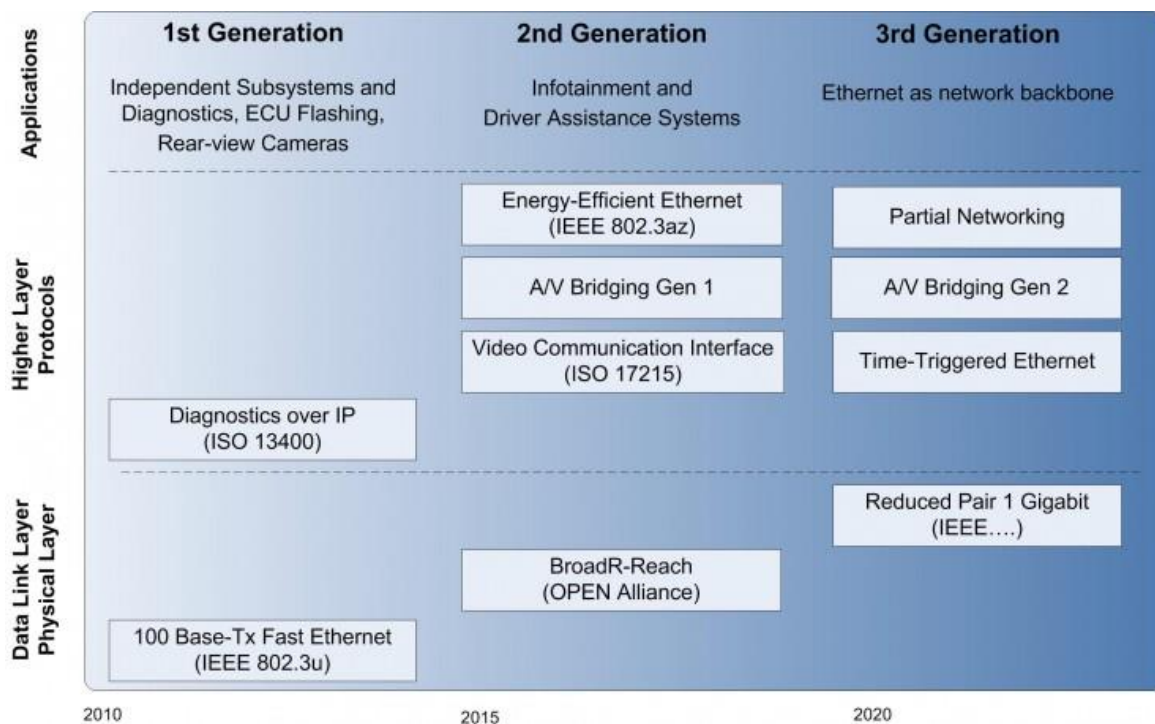


Figure 2.9 Roll-out strategy for Automotive Ethernet described by Hank, Suermann and Müller (2012)

Finally the other key approach adopted within the automotive industry, noted by Mössinger (2010) and Heinecke (2010) among others, is model based development methods. To date the main focus of this has been at a functional level to support the auto-coding of software from models but is now starting to be used at higher levels to model systems and architectures which will be further examined in Section 2.9.

2.5 Agile Software Development Methods

If software development is one of the major challenges of coping with complexity for the automotive industry then it is worthwhile considering what lessons can be learned in the wider software development community. One of the key developments in this field is that of the practice of agile software development. Highsmith, one of the founding fathers and leading proponents of agile software techniques, defines agility as “*the ability to both create and respond to change in order to profit in a turbulent business environment*” and “*the ability to balance flexibility and stability*” Highsmith (2004).

Larman (2004) describes how software developers have learned that software development is subject to emergent behaviour and have adopted agile and iterative development techniques. An early software development model was the waterfall model whereby the outputs of one stage cascade to the next stage and the outputs are developed in a linear sequential fashion. This model has been augmented to add a mirror image test and validation stream in the V-Model. This model also has been adopted on a wider basis e.g. vehicle product development system is often based on a V-Model. However in practice there are issues in implementing sequential models as projects become more complex and difficult to predict. The key learning from the software development community from the widespread

adoption of agile practices for successfully delivering complex products is to conduct iterative and incremental development which leads to progressive integration and testing.

2.6 Quality methods

This section reviews “traditional” quality methods used within automotive industry to consider their applicability to robustness of complex electronic systems.

2.6.1 Failure Mode Avoidance (FMA)

The standard approach within automotive to eliminating failure modes is through Failure Modes and Effects Analysis (FMEA) which is formalised in SAE standard J1739 (SAE 2009). FMEAs have traditionally been reliability focused and have dealt with hardware failure modes which has limited use in the context of robustness which encompasses variations and non-transient events.

Campean and Henshall (2012) recognise the challenges of complexity in automotive electronics and advocate a combination of failure mode avoidance (FMA) based on quality techniques such as FMEA, interface matrix, P diagram, boundary diagram, with systems engineering. They propose an approach for failure mode avoidance with increased emphasis on function and specific attention to interfaces. Much of their proposed methodology is good practice and inherent in systems engineering. For example the authors introduce techniques for functional analysis e.g. System State Flow Diagram as new techniques, however fundamentally these appear similar to existing diagrams within established modelling languages, such as the Systems Modelling Language (SysML), which are already relatively mature with supporting toolsets.

From complex systems perspective the biggest concern is that the methodology proposed by Campean and Henshall does not appear to take into account emergent behaviour with their stated FMA paradigm vision of “right first time through design” and reliance on desktop analysis methods. This is counter to the experience from agile software development methods which recognise that software development is inherently “discoverative”, i.e. some aspects of the design will only become apparent during its implementation, and the need for an incremental iterative approach. A more realistic approach for complex systems is failure mode discovery not failure mode avoidance. Failure mode avoidance is useful but insufficient for complex systems where it is not possible to anticipate failure modes without actually implementing the design.

Recent work by Johannesson, Bergman, Svensson, Arvidsson, Lönnqvist, Barone, and Maré, (2013) has suggested methods of extending FMEA to incorporate robustness issues notably through tools for handling failures due to variation. They discuss uncertainties in design due to noise factors effecting the system but also in terms of uncertainty due to lack of knowledge in the fidelity of a model or of true customer usage patterns. They suggest the use of Ishikawa (fishbone) diagrams to investigate the sources of variation while noting the importance of a thorough understanding of the problem area. They then propose a Variation Mode and Effects Analysis (VMEA) which can be repeated through the design phases as understanding of noise factors increases. The VMEA consists of understanding the sensitivity of key product characteristics (KPCs) to noise factors (NFs), this is done initially by expert judgment firstly by a rating on a coarse scale then a wider scale they term a sensitivity fan. Subsequently when analytical or numerical relationships are known then a “probabilistic VMEA” could be carried out. The focus of the methodology seems to be purely mechanical systems and it is not clear how this would take into account more complex interrelationships, large scale systems or discontinuous nature of electronic or software based systems.

2.6.2 Design for Six Sigma

Yang and El-Haik (2003) give a comprehensive overview of Design for Six Sigma (DFSS). They state that the ultimate goal of DFSS is to “*design it right the first time*” by anticipating the effect of conceptual or operational design vulnerabilities. Again this assumption of designing it right first time is in conflict with the property of emergence found within complex systems.

Yang and El-Haik attribute conceptual vulnerabilities to the violation of design axioms and principles. The first axiom is to maintain the independence of the functional requirements to reduce coupling, i.e. to ensure design parameters can be changed while only affecting a single functional requirements. The second axiom is that the best design should be the one with the least information content. The less information specified to manufacture or produce the design, the less complex it is. While these axioms may work for purely mechanical systems they are implausible for complex electronics and software based systems where they are by definition interdependent and typically suffer from insufficient specification.

Yang and El-Haik attribute operational design vulnerabilities to the lack of robustness in the use environment. A P-diagram is used to understand noise factors and to minimize their impact on the desired output. This is through developing a transfer function in a form of a mathematical or an analytical model to predict the optimum combination of design parameters that minimise the effects of variation. Again applying this approach to more complex electro-mechanical systems would be much more difficult due to the complexity of defining the transfer function.

2.7 Design Methods for Robust Systems

In this section approaches taken in across various domains to designing robust complex systems will be reviewed.

2.7.1 Design principles for complex systems

Schulz and Fricke (1999) identify robustness as one of 4 critical characteristics of a system architecture to support their concept of “*Design For Changeability*”. This is robustness to failures resulting from changes in the systems design and configuration as part of an ability for an architecture to evolve but this property is also important whereby an architecture may have high variant complexity as is typical in automotive. They then identify the basic design axioms that they correlate to robustness in this context as; *Ideality/Simplicity* - systems with only necessary functions and simple interfaces with loose coupling but strong cohesion across modules, *Independence* - such that changes in one part of the system do not require changes in other parts, *modularity/encapsulation* again reflecting need for strong cohesion but loose coupling, as well as *redundancy* and *reliability*.

This need for simplicity in interfaces is consistently stressed by Kopetz. In Kopetz (1999) he argues in order to cope with the complexity of distributed real-time systems it is necessary to decompose them into autonomous elements with simple “*elementary*” interfaces to support “*composability*”. Composability means that a property of the constituent parts e.g. timeliness and accuracy of data, ability to perform a function, of the system will still hold true when the system is integrated. An elementary interface is one which is unidirectional e.g. one node broadcasting out a continuously updated value. A composite interface is one where there has to be an interaction between nodes e.g. handshaking where an acknowledgement is

required or queuing whereby the send has to wait for the receiver to clear their incoming message queue before transmit can be successful.

Kopetz goes on to classify interfaces as; Event triggered, Client Server or Time Triggered. Event triggered interfaces are where an event leads to a single message hence each message must be received and acted on by the receiver according to the temporal order of events. While holding the potential to be fast acting in a simple system, this may not be maintained in a more complex system with many incoming messages. A further evolution of this is the client-server interface where a request response transaction takes place, whereby the initial request is an example of an event triggered interaction hence having the same issues of composability. Time triggered interfaces are of the elementary type whereby a status signal is periodically broadcast to a predetermined schedule which the receivers monitor and take action when a specific condition becomes true. In this case the control flow is unidirectional and provided the update rate is adequate is fully composable. From the author's experience all types are used within automotive. The tendency within CAN networks is to use time triggered approach or a hybrid of events and time triggered whereby a change in status of a broadcast signal is indicated by an update bit. This means a receiver has only to continually monitor the update bit. However within MOST infotainment networks the control is based on a composite approach, the implications of this are examined in Chapter 4.

2.7.2 Fault tolerance design approaches

The "Byzantine Generals Problem" was used in a landmark paper by Lamport, Shostak, and Pease (1982) to illustrate issues of fault tolerance in computer systems whereby in the presence of one or more malfunctioning parts (or traitorous generals in the illustration) algorithms must be used to ensure the properly functioning parts of the system can reach agreement. The original paper postulated solutions for getting correct agreement but noting

that this is only achieved with inherently expensive solutions, in terms of time and resources, unless assumptions are made about the type of failure.

Some 20 years later Driscoll, Hall, Sivencrona, and Zumsteg (2003) review Byzantine failure properties and “difficulties” in approaches to fault tolerance from a practitioner’s perspective. They state that humans have difficulty in relating to and understanding the significance of extremely low incidence faults and incorrectly assuming them to occur with zero or low probability, whereas in fact such problems can be caused by many known failure mechanisms. They point out studies which suggest with increasing speed and decreasing geometries the reliability of Commercial Off The Shelf (COTS) integrated circuits is decreasing. They also highlight increasing use of distributed systems for high integrity applications, such as automotive "X by wire", where consensus is required across the system. Both of these factors increase the need to avoid Byzantine failures where consensus actions are required yet industry has apparently not given as much consideration as academia.

Driscoll et al. (2003) give a typical example of a Byzantine problem as a logic signal stuck or during an extended rise time being at an indeterminate voltage level whereby multiple receivers can interpret the signal differently depending on their own thresholds. Another example is an event that occurs exactly on its deadline which is seen as some observers as acceptable but not by others due to very small differences in clock synchronisation. A further example was a badly terminated signal bus causing reflections which led to different signals depending whether receivers were at a node or an anti-node.

They also give an example whereby one type of aircraft was nearly grounded despite being designed with high levels of redundancy as an unforeseen systematic fault occurred in the supposedly independent elements. They note the use of Time Triggered communications protocols developed for automotive which allow synchronisation of nodes and deterministic communications, gaining acceptance within aerospace but have concerns this leaves the

protocol itself vulnerable to Byzantine failures citing tests where fault injection has caused individual nodes in a system to divide into two cliques.

They also note that a common fallacy is to assume low incidence faults occur uniformly distributed in time whereas in practice the precipitating events may occur at specific times which will cause them to be clustered.

They identify 3 generic approaches illustrated with examples to dealing with Byzantine failures. The first approach is “*Full Exchange*”, as used in SPIDER (Scalable Processor Independent Design for Electromagnetic Resilience), which is based on the approach suggested in the original Byzantine Generals paper with peer to peer communications between all nodes. The second approach is “*Hierarchical Exchange*”, used in Honeywell's SAFEbus, which uses private exchanges within sub-sets of nodes then exchanges between the sub-sets. The final approach is “*Filtering*” used in TTP (Time Triggered Protocol) star topology whereby a truly independent bus guardian is added to filter any asymmetric manifestations of known types of Byzantine fault, such as delayed signals. They note that a form of filtering is also used in other approaches by discounting nodes from subsequent rounds of exchanges.

Driscoll et al (2003) make a final statement on practical approaches to Byzantine failures as follows, “*Anyone designing a system with high dependability requirements must thoroughly understand these failures and how to combat them.*”

Arora and Kulkarni (1998) seek to simplify achieving fault tolerance by splitting the task between two classes of components one termed “Detectors” and the other “correctors”. Detectors are system components which ascertain whether a system is working correctly and include error detection codes, watchdogs and alarms. Correctors are system components which attempt to bring system back to a correct state including error correction codes, voters and reset procedures.

In Powell (2011) the results of a collaborative research project to develop a Generic Fault-Tolerant Architecture for Real-Time Dependable Systems by aerospace, military and rail partners to bring down cost of fault tolerance are discussed. However in common with the methods previously reviewed they still rely heavily on redundancy at a level not affordable for non-critical functions in a cost-sensitive product.

2.7.3 Fault recovery approaches

Arshad, Heimbigner, and Wolf (2004) describe a planning-based approach to recovery from failure of a distributed system in which the effects of a failure are assessed and one of a number of approaches to recovery are dynamically selected. This approach is targeted at large scale IT networks and seems too complex and potentially time consuming for an embedded system, where having a single default recovery path for each node and levels of immunity or graceful degradation to loss of other nodes seems a more practical approach. However the approach to failure modelling, whereby dependencies to a failure in the system are modelled as failed, affected or normal, could be of use as part of an offline analysis of the robustness of a design to failure - particularly where there is potential for cascade failures.

Work on the “*survivability*” of large scale critical system architectures under the name of the Willow Architecture is described by Knight, Heimbigner, Wolf, Carzaniga, Hill, Devanbu and Gertz (2001). This is aimed at critical national and defence architectures and features a range of mechanisms to deal with faults (including attacks) encompassing fault prevention, fault tolerance and fault removal through redundancy, fault monitoring and proactive and reactive reconfiguration.

Similar approaches have been proposed at a vehicle level notably by Amor-Segan, McMurrin, Dhadyalla, and Jones (2007) as part of a “*Self Healing Vehicle*”. While such approaches for reconfiguration have not yet been applied in any production vehicle to the

authors knowledge, the concepts of having a standard operating environment in multiple ECUs being pursued in AUTOSAR and the increase in processing power of automotive microprocessors and in bandwidth of bus systems such as Flexray make this an increasingly viable concept.

Kopetz (2011, p14-15) discusses “*fail-safe real-time systems*” whereby a safe state can be reached sufficiently quickly in the event of an error being detected and “*fail operational real-time systems*” whereby a minimum level of service is required to avoid a catastrophic event e.g. an aircraft flight control system. For a fail-safe system to work the error detection must have very high reliability and is often achieved by the use of watchdog devices which will invoke the fail-safe state if a periodic heartbeat signal is not received. For fail-operational systems redundancy is required to maintain the level of service in event of a failure.

2.7.4 Self Stabilization

A good overview of self-stabilization is given by Dolev (2000). A distributed system can be said to be self-stabilizing if it returns to known good state after starting in an arbitrary state. This arbitrary state could be the result for transient failure such as a power interruption or processor crash or corruption of variables. Hence self-stabilization can be considered as a universal fault recovery mechanism. There is also a weaker form of the concept called Pseudo-self-stabilization whereby a system may deviate from legal behaviour a defined number of times before reaching a good state.

An important distinction in designing self-stabilizing algorithms is whether the distributed system is synchronous with some form of universal clock or asynchronous with each node having its own timebase. Self-stabilizing algorithms must run continually and repeatedly communicate with neighbouring processors. For distributed systems consisting of

identical elements randomization can be used to defeat symmetrical properties which may prevent stabilization. There are a large number of algorithms typically aimed at distributed networking management tasks such as leader election and mutual exclusion algorithms and distributed reset (Arora, A., and Gouda, M., 1994).

Self stabilization may have merit in a distributed automotive system context for improving robustness but would come with an overhead in terms of processing and communications to run the necessary algorithms.

2.8 Safety engineering

Kopetz (2011 p14-15) points out that Safety is akin to reliability but in the context of critical failure modes. This section reviews safety engineering practices to determine whether any could be adapted to use in a robustness context.

2.8.1 Safety Standards

In the early 1980s and 90s it became recognised that reliability was a property that could not be added to software or robustly proven to exist at extreme levels required for safety critical systems through validation but must be inherent in the degree of rigour applied to the design and development process (Butler and Finelli 1991, Littlewood and Strigini 1993). This has led to the introduction of standards for the development of safety critical software controlled systems. A generic standard was first published by the International Electrotechnical Commission (IEC) in multiple parts between 1998 and 2000 under the title of “*IEC 61508 Functional safety of Electrical/Electronic /Programmable Electronic Safety-Related systems*” (Brown 2000). IEC61508 uses a risk-based approach for determining the required performance of safety-related systems, requiring a hazard analysis and risk assessment to ensure risk is reduced to a tolerable level. As part of this a Safety Integrity

Level (SIL) is assigned which determines the level of rigour required for the development of software based systems (IEC 2005).

IEC 61508 was always intended as a generic standard which would have sector specific implementations. The automotive industry subsequently developed such a standard ISO26262 which is better adapted to the specific requirements of real-time embedded systems, automotive development and life cycles, manufacturer / supplier relationships and ‘consumer-goods’ (Rau and Weiss 2008).

Both IEC61508 and ISO26262 give recommendations (but are not prescriptive) as to techniques which are suitable for implementing systems at various safety integrity levels. These techniques have been reviewed and categorized into the areas of design methods, design analysis, fault detection and diagnosis, fault tolerance, defensive programming and verification methods.

Design Methods

The first area the standards address is the approach to the design in requiring structured design method and strongly recommending the use of semi-formal methods such as models or state transition diagrams to specify the design. In doing this they reflect much of current design practice. The standards also mention formal methods but only recommend for highest software integrity levels.

They also enforce good practice regarding selection of suitable programming language which supports strong typing and using defined subset of the language. They recommend the use of coding standards, style guidelines and naming conventions for modelling and in-house conventions for documentation maintainability. They also recommend that design tools should be well proven and for higher integrity levels be certified.

The safety standards also echo the design principles for complex systems in requiring a modular hierarchal design with precisely defined simple interfaces that avoid complexity of

hardware and software components where possible. The standards also make the point that the design should be maintainable during service and testable during development and operation. The safety standards also echo the AUTOSAR principles of reuse of a library of trusted and verified software modules and components but with a goal of increasing reliability rather than reducing effort.

Design Analysis Techniques

A major part of the recommendations of the safety standards relate to design and analysis techniques to; identify safety hazards in systems, the possible causes and consequences and to determine actions to prevent or mitigate them. Techniques for failure analysis include; Inductive Analysis techniques such as Failure Modes and Effects Analysis(FMEA), Fault Tree Analysis (FTA), Hazard and Operability Study (HAZOP) and common cause failure analysis of diverse software or redundant hardware.

Other analysis techniques recommended, which could be relevant to robustness, include: reliability block diagrams which model separate conditions which must be fulfilled for successful operation, Markov models which use probability chains to predict reliability, safety or availability of a system and sneak circuit analysis which attempts to identify latent hardware, software or operator actions could cause the system to malfunction under certain conditions.

The use of simulation and modelling is also recommended to ensure the system meets specified requirements including use of Monte Carlo simulations. Monte Carlo simulations can be used to simulate stochastic phenomena such as noise or to produce large samples of data from which statistical results can be obtained.

When changes are made to one part of system, a recommended technique is impact analysis in order to understand the potential effects on the wider system hence the rigour that must be applied to the development and how widespread verification activities have to be.

The safety standards also recommend the use of safety cases which are structured arguments as to why the designers of a system would consider it to be safe for a given context of use linking safety objectives to supporting evidence e.g. test results. This technique could clearly be applied to argue the robustness of a system and is investigated in more detail in the next section.

The standards also recommend the use of design reviews to ensure the design is well structured and documented such that they can be communicated and to systematically engage wider peer and expert input and experience into the design. Again a similar concept of design peer review would be equally applicable for robustness.

Fault detection and diagnosis

The software safety standards recommend a series of mechanisms for error detection. These include; range checks of input and output data, plausibility checks, detection of data errors, external monitoring facility, temporal and logical program sequence monitoring, watch-dog timers, input-output, communication, memory and hardware monitoring, comparators and voters between independent processing units, failure detection by redundant hardware, and reciprocal comparison by software.

A number of the fault detection and diagnosis measures recommended by the safety standards could be implemented at little or no additional piece cost. However, noting Kopetz's (2011) comments, these must be implemented to a high degree of reliability if they are not to make the system less robust through false positive detections.

Fault Tolerance

The safety standards recommend mechanisms for error handling including; static recovery, graceful degradation, independent parallel redundancy and correcting codes for data.

Defensive programming

They also recommend the use of defensive programming techniques which detect anomalous control flow, data flow or data values during program execution and react to these in a predetermined and acceptable manner. Specific defensive programming techniques they recommend include; failure assertion programming, diverse programming, recovery blocks, backward and forward recovery and retry fault recovery mechanisms.

Verification Methods

The safety standards also recommend a wide range of verification techniques beyond requirements based functional testing. A major theme of interest for robustness is that of black box interface testing whereby input data is selected to exercise a statistical coverage of all specified input cases, error cases boundary cases, including dynamic analysis and testing. Of particular note are the methods recommended for deriving test cases which include; analysis of requirements, analysis of external and internal interfaces, generation and analysis of equivalence classes for hardware-software integration, analysis of boundary values, error guessing based on knowledge or experience, analysis of functional dependences, analysis of environmental conditions and operational use cases and analysis of field experience.

The need for domain knowledge is particularly noted for fault insertion testing and error guessing where looking for emergent behaviour. This is well stated in part of ISO26262 as follows "*field experience, and testing experience and intuition combined with knowledge and curiosity about the system under test may add some uncategorized test cases to the designed test case set*".

Other robustness relevant tests are stress tests which "*verifies the test object for correct operation under high operational loads or high demands from the environment*" and model-based testing. For model-based testing they recommend back to back comparison test between model and code whereby the model can be used as a test oracle.

For the highest levels of safety criticality the use of formal methods to prove the correctness of program or specification mathematically rather than empirical verification is recommended as a possible technique. Formal methods will be examined in Section 2.11.

2.8.2 Safety cases

The developers of safety critical systems have to justify that their system is acceptably safe to relevant stakeholders including regulators and potentially in legal proceedings through a “safety case”. A safety case has been defined in following terms “*A safety case should communicate a clear, comprehensive and defensible argument that a particular system is acceptably safe to operate in a particular context*” (Kelly and Weaver 2004). The safety case should clearly argue how evidence provided, e.g. test results or specifications, fulfills the safety requirements of the system resulting in a system that is acceptably safe. Kelly and Weaver make the observation that safety arguments are frequently badly explained and ambiguous. They propose the use of a diagrammatic technique known as Goal Structuring Notation (GSN) as a more clear and structured method of making safety arguments. GSN has symbols representing Goals, Solutions, Strategies, Contexts, Assumptions and Justifications with arrows representing relationships. Hence a goal can be seen in a particular context to be solved by particular solutions for which evidence exists. These safety arguments are constructed in a hierarchical form and strategies can decompose a high level goal into sub-goals.

Proven strategies and solutions to meeting safety goals can be re-used. An issue with re-use is to ensure the solution is not simply cut and pasted to a context in which it may no longer be appropriate - Kelly and McDermid (1997) refer to this as “*applicability*”. Kelly and McDermid show how generic arguments can be constructed as “*patterns*” in GSN so can support and systematize re-use while ensuring that the rationale behind an argument and the

context are clearly understood. This notion of safety case patterns has been subsequently extended to illustrate flawed or weak arguments which should be avoided – so called “*anti-patterns*” (Kelly and Weaver 2004).

Modular approaches to developing safety cases using GSN have been developed (Kelly 2001, Bates, Bate, Hawkins, Kelly, and McDermid, 2003). The intent of this work is to allow safety cases to support increasing use of modular architectures within avionics which may be used in different configurations and up-graded. This work concludes that the safety case has to also have a well-constructed architecture with defined partitioning in order to reason independently about different parts of the system and allow a composition of safety cases as a series of modules.

The concept of extending the safety case to cover dependability has been examined by Despotou and Kelly (2004). Dependability as well as safety also encompasses aspects such as security, maintainability and availability and they highlight that these aspects may be in conflict and so there may be a need to trade-off between them. They show that it is possible to view this as a modular argument using GSN with the top level dependability argument being composed of arguments for the individual attributes plus an argument for the trade-offs between them which relies on a trade-off methodology.

Habli, Ibarra, Rivett, and Kelly (2010) propose methods of how to integrate the development of a safety case with model based development, illustrating it with a case study of an automotive air suspension system. Specifically they consider how a GSN safety case can be integrated with a SysML model being used in a model based design process, pointing out that since both SysML and GSN are diagrammatic, hierarchical and modular techniques they are both compatible and can allow the analysis of complex systems to be broken down into more manageable chunks.

Habli, Hawkins, and Kelly (2010) point out that safety assurance of software is reliant on the processes used to develop the software but contest that there needs to be a more explicitly defined argument linking the processes to the software assurance required for a particular function as not all functions presenting the same degree of risk and hence require the same rigour. They propose linking the process based argument to the product argument including factors such as tool and method integrity, competence of personnel, and configuration management.

An interesting question the authors pose in their conclusions is whether for complex systems which can suffer systematic failures in the hardware integration - is there a need to move from an integrity based argument (typically based on failure rates of constituent parts) to an assurance argument addressing the potential sources of systematic failures e.g. the development processes employed? Their conclusion is that wherever there is complexity regardless of implementation or process there is a need for an assurance argument.

Safety cases are a systematic, yet flexible method which could potentially be applied to the analysis of how a system can be argued to be robust encompassing various facets including design features and design methods.

2.9 Modelling Complex Systems

The ability to model complex systems is both a key challenge and enabler for their successful development. Kopetz (2011) states that *“the major challenge of design is the building of a software/hardware artifact that provides the intended behaviour (i.e. the service) under given constraints and where relevant properties of this artifact can be modelled at different levels of abstraction by models of adequate simplicity”*. This is an echo of a much earlier quote by Lord Kelvin (Thom1904) *“I am never content until I have*

constructed a mechanical model of the subject I am studying. If I succeed in making one, I understand; otherwise I do not”.

Simplification strategies Kopetz advocates are; abstraction, partitioning, isolation by suppressing irrelevant detail and segmentation referring to the temporal decomposition of behaviour in into sequential parts. He goes on to give a useful definition of a model as “*a deliberate simplification of reality that is relevant for a particular purpose*”.

While it is impossible to close the lid on the Pandora's box of increasing functional requirements, can complex systems of systems be usefully modeled at a level of abstraction that allows robustness properties to be examined? This section will review two major areas of on-going work in the field of modelling complex systems. Firstly architectural modelling approaches will be examined both automotive specific and more generally applicable work on system of systems. Secondly co-simulation approaches will be reviewed which attempt to make a dynamic model on a large scale system by linking concurrent simulations of constituent parts.

2.9.1 Architectural Modelling Approaches

2.9.1.1 EAST-ADL

The major body of work in developing a system level model approach to automotive electronics has been the EAST-ADL which has been developed as a standardized approach through a series of European collaborative projects. There is extensive literature from the project but a good overview can be derived from the ATESSST2 Project Brochure (ATESST2 2010) which describes the status at the end of the project. Ongoing information is also available from the EAST-ADL Association website (<http://east-adl.info>) and from the website of the current MAENAD project (<http://www.maenad.eu>).

EAST ADL is an approach to automotive system modelling based on an architecture description language (ADL) that keeps information in a single data structure. The claimed benefits are achieved in improved development time, cost efficiency, quality and dependability. EAST-ADL was initially developed in the EAST EEA and more recently further refined in the ATESSST project including compatibility with newer standards such as AUTOSAR and increased dependability analysis.

The scope of EAST ADL is functions, hardware and software for automotive embedded systems and environment. This defines architectural elements and a number of views that give useful abstractions of the overall system for specific purposes during system development lifecycle. Figure 2.10 illustrates the EAST ADL abstraction levels and extensions. Specific use cases include; feature modelling, variant analysis, environment modelling, structural and behavioural modelling of software and hardware, requirements modelling including traceability, timing and failure analysis.

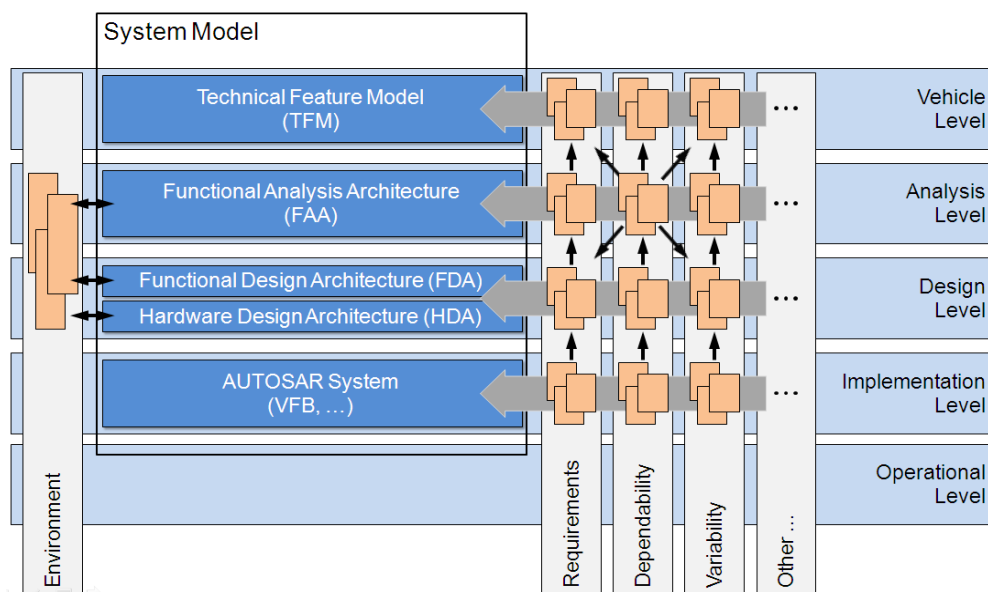


Figure 2.10 EAST ADL abstraction levels and extensions (East ADL association website)

The vehicle level represents an implementation independent model of vehicle feature content and properties. The analysis level contains a more detailed breakdown of functionality, but still abstracting hardware and software deployment, including interactions between functions. The design level then maps the features and functions to hardware and software architectures taking into account issues such as re-use of carryover components and use of off-the-shelf standard design elements. The implementation level details the software architecture at component level for basic software, standard software functions and application software in an AUTOSAR compliant manner.

The modelling approach used in the EAST-ADL language is through a meta-model which defines stereotypes for model entities which are implemented as a UML2 profile. This profile can then be used to create model diagrams in a UML tool with stereotyped entities such as vehicle features which already have defined property types. While EAST-ADL is based around UML2 it also makes use of some of the extensions found in SysML, which is in itself a UML2 profile, such as for requirements traceability. The models give the overall system architecture including interrelationship between functions but a function's internal behaviour is typically defined by other techniques.

EAST-ADL can support timing analysis between top down timing constraints of the functions and the bottom up properties of the implementation utilising the AUTOSAR Timing Extension. Timing requirements such as end to end delay or synchronicity can be modelled as constraints on event chains as illustrated in Figure 2.11.

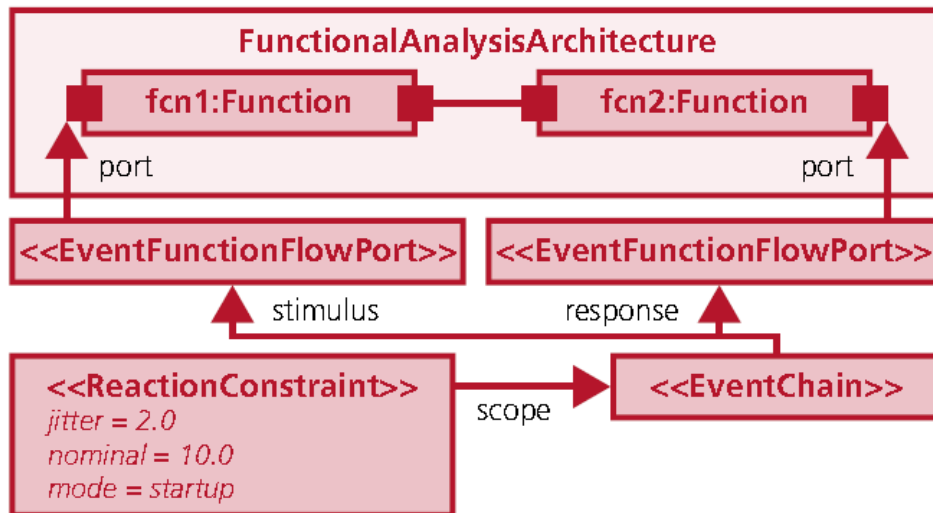


Figure 2.11. EAST-ADL Event Chain with associated timing constraint (ATESST2 2010)

EAST-ADL supports requirements traceability and decomposition including those of system properties which may be safety relevant, hence it can facilitate ISO26262 processes. EAST-ADL model levels can be mapped on to ISO26262 process requirements and EAST-ADL has defined a dependency package to support analysis such as ASIL (Automotive Safety Integrity Level) decomposition and error modelling. The EAST-ADL error model describes behaviour during a fault condition and attempts to show how this could propagate throughout the system using explicit error propagation ports and connections (see Figure 2.12). These information interdependencies within the system can be used by external tools for safety analysis such as HiP-HOPS tool (Hierarchically Performed Hazard Origin and Propagation Studies) for static safety analysis in terms of FFA, FTA, and FMEA.

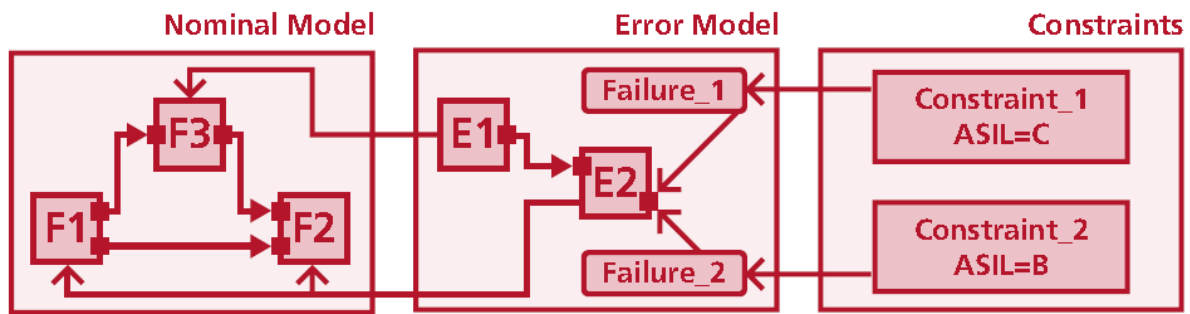


Figure 2.12. EAST-ADL error model (ATESST2 2010)

This illustrates a key point regarding EAST-ADL that is fundamentally a structural model showing entities and their relationships does not attempt to contain the behavioural properties of functions which require other modelling tools such as Simulink or Modelica. It is not apparent nor claimed that this will identify any form of emergent behaviour without any use of behavioural diagrams such as state charts, activity diagrams, message sequence charts. However by defining the relationships between behavioural models EAST-ADL can provide a structure to support the integration of multiple behavioural models for analysis at a system or system of systems level. Section 2.11 on formal methods will review work to use EAST-ADL to facilitate formal analysis techniques. To enable linkage between tools EAST-ADL has implemented an Eclipse-based tool platform centered around a UML2 modelling and profiling environment called Papyrus UML.

2.9.1.2 COMPASS

Fitzgerald, Larsen, and Woodcock (2014) give an overview of the vision of the COMPASS project in creating a collaborative development environment for the development of Systems of Systems. Their approach, illustrated in Figure 2.13, is based on an interlinked tool architecture comprising of; a SysML modelling tool (Artisan Studio) for modelling the system of systems architecture, a platform for developing more detailed models and conducting static analysis including formal methods and dynamic simulation and analysis (COMPASS Overture) and a real-time testing platform (RT-Tester) for the development of test cases and conducting automated testing.

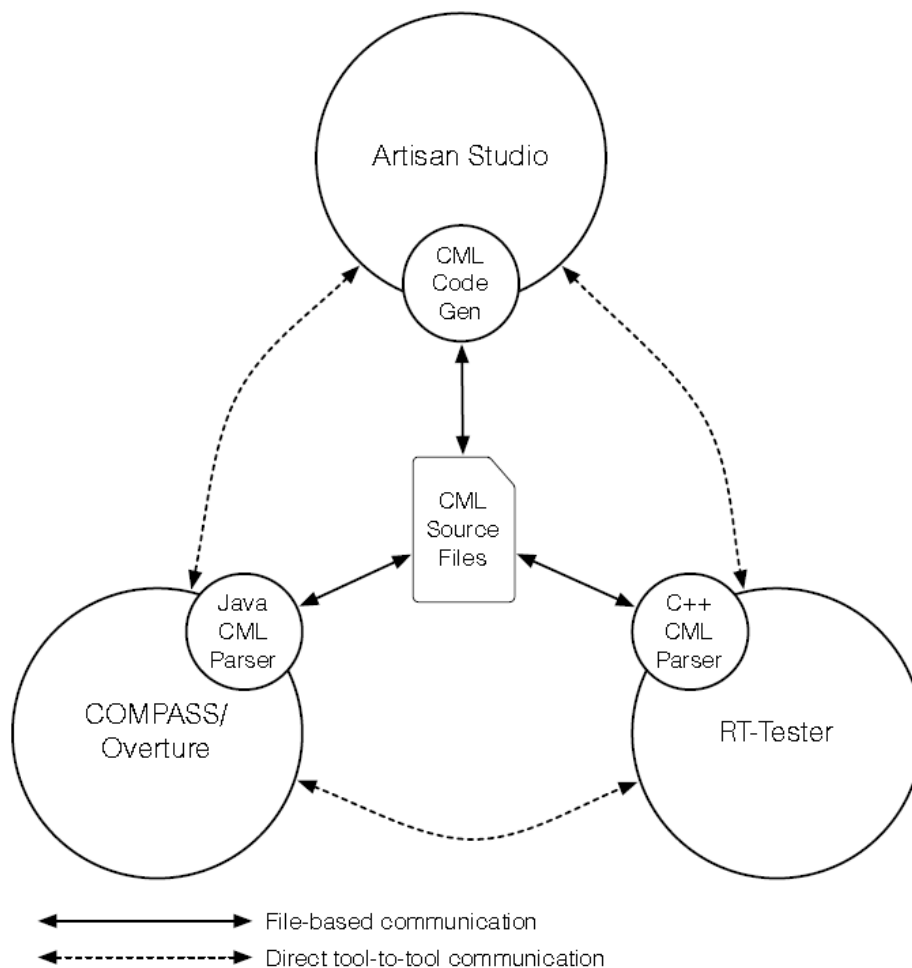


Figure 2.13 Overview of the COMPASS toolset (Fitzgerald, Larsen, and Woodcock, 2014)

To facilitate the modelling of system of systems and exchange of data between tools the project will develop a modelling language called the COMPASS Modelling Language (CML). CML is based on experience of VDM (Vienna Development Method) state based formal method and the CSP/Circus process-based formal methods. The project also aims to support fault modelling and analysis at both an architectural level for the whole system of systems and detailed level for constituent systems.

Andrews, Ingram, Payne, Romanovsky, Holt and Perry (2014) describe the project's approach to fault modelling using a set of SysML diagrams. This fault modelling profile uses structural diagrams (block definition diagrams) and behavioural diagrams (activity and sequence diagrams), to give views on nominal behaviour, fault activation, erroneous behaviour, fault tolerance and recovery mechanisms. These support understanding of the fault behaviour and the definition of the recovery mechanisms. The models are not executable but future work is to develop semi-automatic translation to the CML to allow formal verification of dependability related properties of fault tolerance models.

To address issues of confidentiality in sharing models of constituent systems, a contract approach is being developed to develop abstracted behavioural formal models of the system interface (Bryans, Fitzgerald, Payne and Kristensen, 2014). The project's strategy to tackle the issues associated with testing large scale models is to use knowledge gained from testing the individual systems.

2.9.2 Co-Simulation Modelling Approaches

2.9.2.1 DEST ECS

Broenink, Kleijn, Larsen, Jovanovic, Verhoef and Pierce (2010) describe the goals of a European collaborative project called DEST ECS (Design support and tooling for dependable embedded control software). The DEST ECS project advocates a collaborative and

multidisciplinary approach to development of models of embedded systems and its major output is an Integrated Development Environment (IDE) to support this. The approach to creating this IDE is through a co-simulation of a discrete time VDM (Vienna Development Method) model of the control system and a continuous time model of the plant using a physical modelling tool called 20-sim. VDM was selected over the rather more obvious choice of Matlab Simulink as it more directly supports formal analysis. 20-sim is a product developed by one of the partners but they claim similar functionality to state of the art physical modelling tools based on the Modelica language such as Dymola. A key development within the project is a "tool connector" which enables the co-simulation as well as being responsible for model analysis such as fault scenarios and regression testing. The DESTECS approach is illustrated in Figure 2.14.

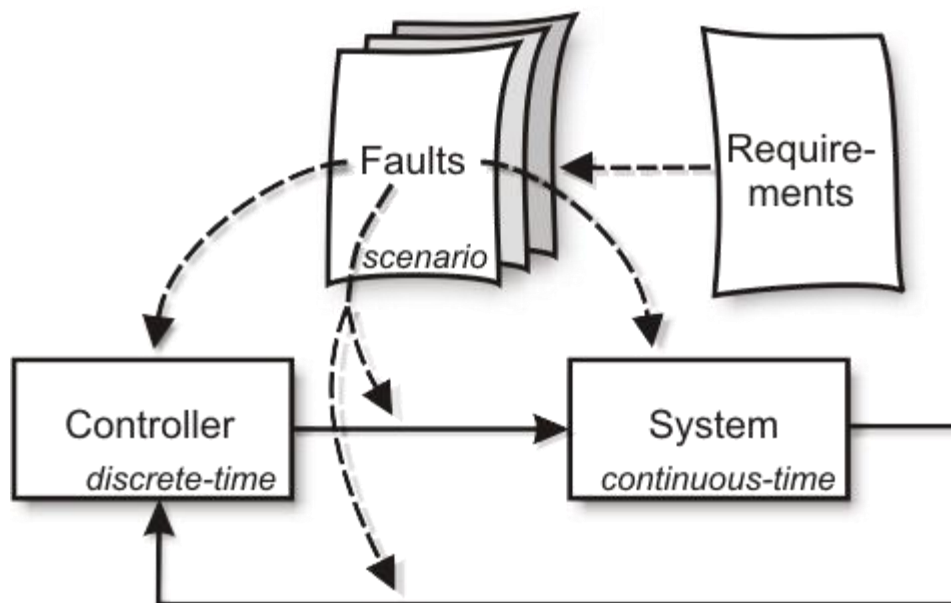


Figure 2.14 DESTECS co-simulation approach to modelling embedded systems (Broenink et al, 2010)

Fitzgerald, Larsen, Pierce and Verhoef (2013) describe the approach to co-model construction is explained as firstly developing a fully discrete event model to gain confidence in the control approach which is subsequently replaced by a more accurate continuous time model. A contractual approach to the interface between the discrete event and the continuous time model is taken defining the nature of communication between these two models. The exchange between the two simulations is intended to be limited to shared time base, variables and events in order to minimize impact on models running in their own optimized environments. Part of the work of the project was to model abnormal behaviours and countermeasures in order to test and improve the dependability of the control system.

2.9.2.2 Functional Mock-up Interface (FMI)

The Functional Mock-up Interface (FMI) standard to enable the exchange of dynamic models and co-simulation is described by Blochwitz et al (2012). FMI was initially developed in a European collaborative project named MODELISAR producing a first version of the standard in 2010 which the authors state is supported by more than 30 tools. The project was initiated by Daimler AG to enable integration and exchange simulation data from a number of different suppliers using different toolsets see Figure 2.15.

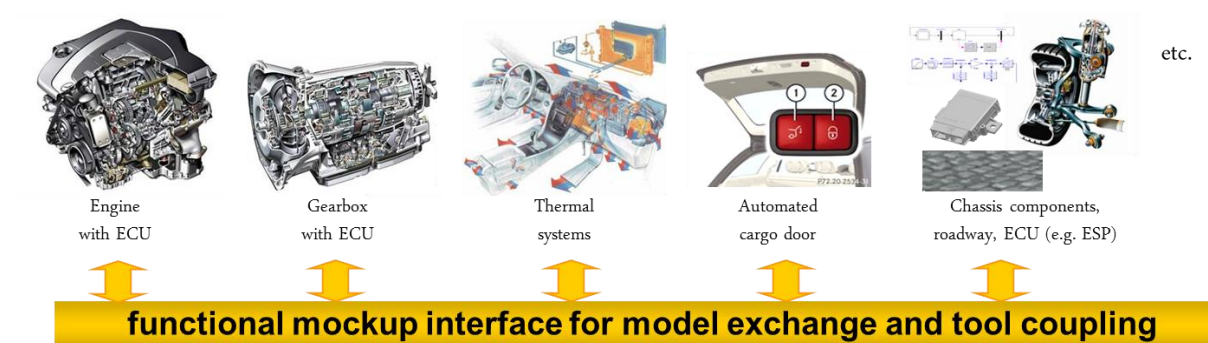


Figure 2.15 FMI Concept for Model Exchange and Co-Simulation (Blochwitz et al 2012).

The model exchange is enabled by a modelling environment generating a C code version of their model that can be imported and used in other modelling environments. Co-simulation is achieved through Master algorithms that control data exchange of explicitly defined variables and synchronisation of slave simulations, including their solvers, through a set of C functions.

Blochwitz et al (2012) describe the main areas of improvement for the second version of FMI which include; explicitly defining the causality of exchange variables, allowing variables to be externally tuned during simulation, the ability to restart a simulation from a saved state, improved time event handling and unit definition. A number of examples of the use of FMI are given, such as software in the loop simulation for transmission controllers at Mercedes-Benz, but none yet at a system of systems level.

2.10 Robustness Test Methods

This section reviews literature on robustness testing which is a subset of non-functional testing. A fundamental issue in this type of testing is experimental design for testing large scale systems with many variables. George Box (2000), a renowned academic in the field of experimental design, argues for an iterative exploration and discovery experimental approach where results of tests are available in a short period of time, particularly where there are a large number of experimental variables. He describes an iterative learning approach to experimentation and identifies six possible options for modification of experiment after initial test. These are to; move to a new location, add another fraction, rescale, drop and add factors, replicate or augment.

A systematic approach to black-box test case design initially developed by Grochtmann, Grimm and Wegener (1993) is the Classification Tree Method (CTM). The

CTM method is a type of partition testing whereby important input factors are classified and further broken down into subclasses. These can be represented in a tree structure similar to a UML class diagram as illustrated in Figure 2.16 and test cases are shown graphically as the intersection nodes of the horizontal lines with selected classes.

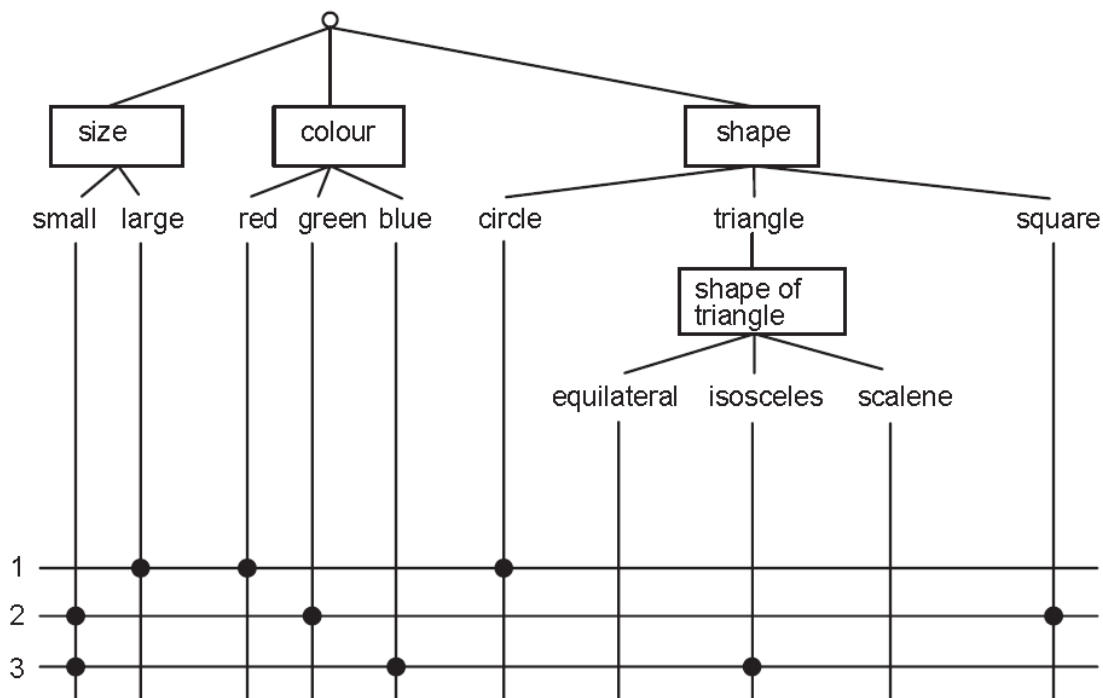


Figure 2.16 Classification Tree (Grochtmann, Grimm, and Wegener, 1993)

The advantages of this method are that it can give a stepwise, graphical mechanism for defining large-scale automated tests which can be supported using a Classification-Tree Editor (CTE) tool. Through intensive industrial application CTE tool has been extended to; avoid selection of invalid test cases, be able to set different priorities levels to classifications and to generate automated test scripts (Lehmann and Wegener, 2000).

However CTM is concerned with input states and does not directly address an important factor in embedded systems of dynamic behaviour. Lehmann (2000) proposes a method for testing dynamic functional behaviour entitled Time Partition Testing (TPT). TPT

breaks down dynamic behaviour of a system into a number of discrete elements, e.g. initialisation, which are called testlets. Testlets define input and expected output behaviour of system over a time series which may have a definite span or continue to run indefinitely. By assembling a series of testlets paths of execution by the system can be defined from initialisation to shut down which can be used as test cases for the dynamic behaviour of the system in an automated testing environment. Bringmann and Krämer (2006) describe the further development of the TPT method through the development of a test language used to formally defined testlets and by the use of classification tree method and editor as a means of graphically defining test cases through assembling a series of variants of testlets as illustrated in Figure 2.17.

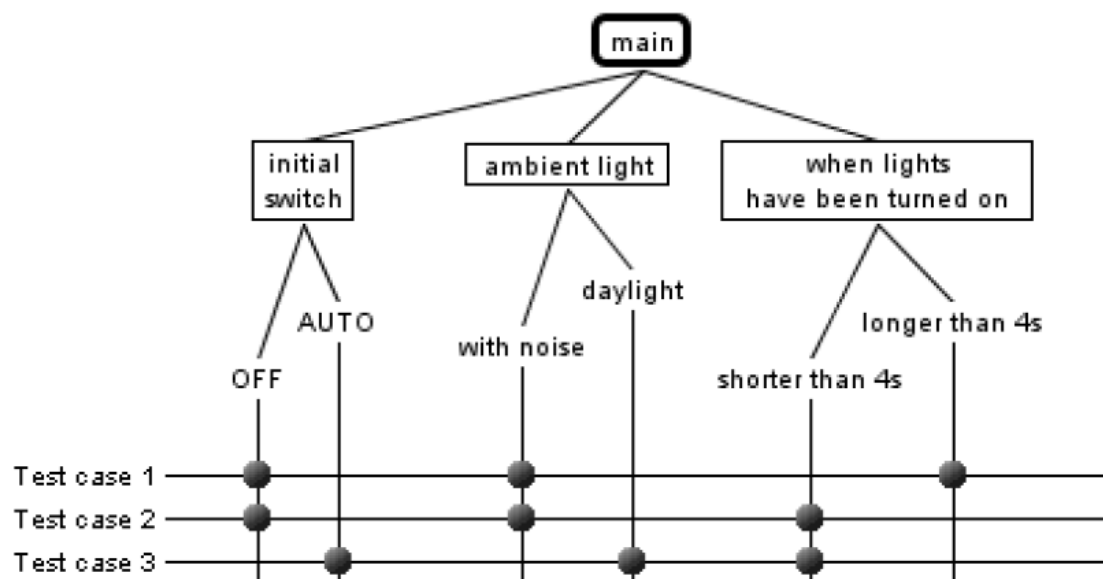


Figure 2.17 Use of Classification Trees to define TPT test cases (Bringmann and Krämer, 2006)

Bringmann and Krämer (2006) claim the unique feature of Time Partition Testing is the way test cases for continuous behaviour are modeled and systematically selected. A further but important benefit they note is that the method is capable of being reactive, i.e. modifying

the test case execution dependent on the behaviour of the device under test. They describe the overall TPT process from requirements to test report, shown in Figure 2.18, and state that it is being successfully used by DaimlerCrysler and several of their suppliers.

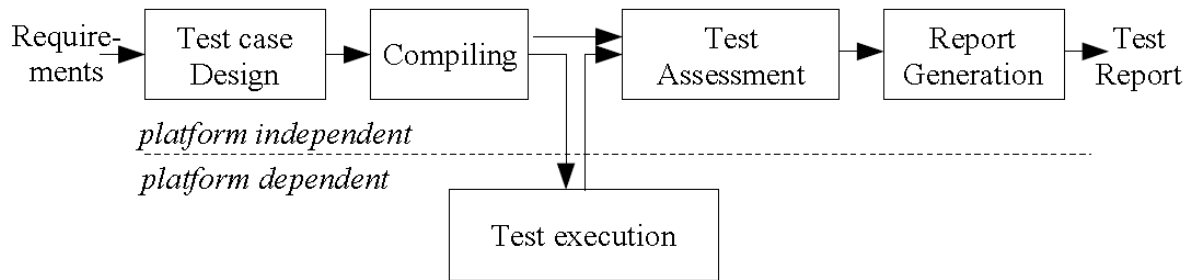


Figure 2.18 TPT test process (Bringmann and Krämer, 2006)

While TPT offers a systematic approach to testing of continuous systems it does not explicitly include the discoverative iterative approach recommended by George Box (2000). Afzal, Torkar and Feldt (2009) review search based techniques, which use meta heuristics to search test space for levels of fitness against a given test criteria, known as Search-based Software Testing (SBST), specifically focusing on application to non-functional testing. The non-functional properties they covered specifically in the search were usability, safety, robustness, capacity, integrity, efficiency, reliability, maintainability, testability, flexibility, reusability, portability, interoperability, security, performance, availability and scalability. Of interest to this research is that robustness was not the focus of any of the work although clearly execution time can be a factor in robustness related failures. One reason may be that the SBST method is not well suited as it may be difficult to define a suitable fitness function for this characteristic which gives some level of continuous indication of robustness. Another reason may well be that this survey was focused on software testing while it may be best to consider robustness at a higher level of testing which includes more factors such as hardware

variations or system interactions that can cause robustness failures rather than just the software.

More recently work specifically focused on robustness testing of automotive electronics against the effects of low voltage transients is reported by Dhadyalla, McMurran, Amor-Segan, Li, et al. (2010). The approach described uses weighted pseudo-randomly generated test sequences to achieve balance of coverage of complete test space and focus on areas where experience has shown faults most likely to occur while allowing automated testing within a reasonable period of time. The approach described supports component, system, and system of systems (full vehicle) testing but is limited to a single critical, variable of supply voltage. The presence of Diagnostic Trouble Codes (DTCs) was used as a fitness function.

Becci, Dhadyalla, Mouzakitis, Marco and Moore (2013) expand on this approach to cover higher number of interacting factors through the use of Sequence Covering Arrays (SCA) in an approach they entitle Non-Functional Sequence Testing (NFST). This approach recognises the importance not just of the order of events in real-time system but also their relative timing. In the NFST approach, illustrated in Figure 2.19, sequence covering arrays are used to generate efficient design of experiments covering all orders of events and assigns a time vector between events which is subject to pseudorandom variation according to an inverse exponential distribution. The test which is conducted on a Hardware In Loop test environment consists of a system initialisation and initial event followed by an NFST sequence of events at pseudo-randomly varied intervals, followed by a “prove-out” test. The prove-out test determines whether the random sequences were able to alter the system functionality, i.e. has the system been able to recover to known good behaviour.

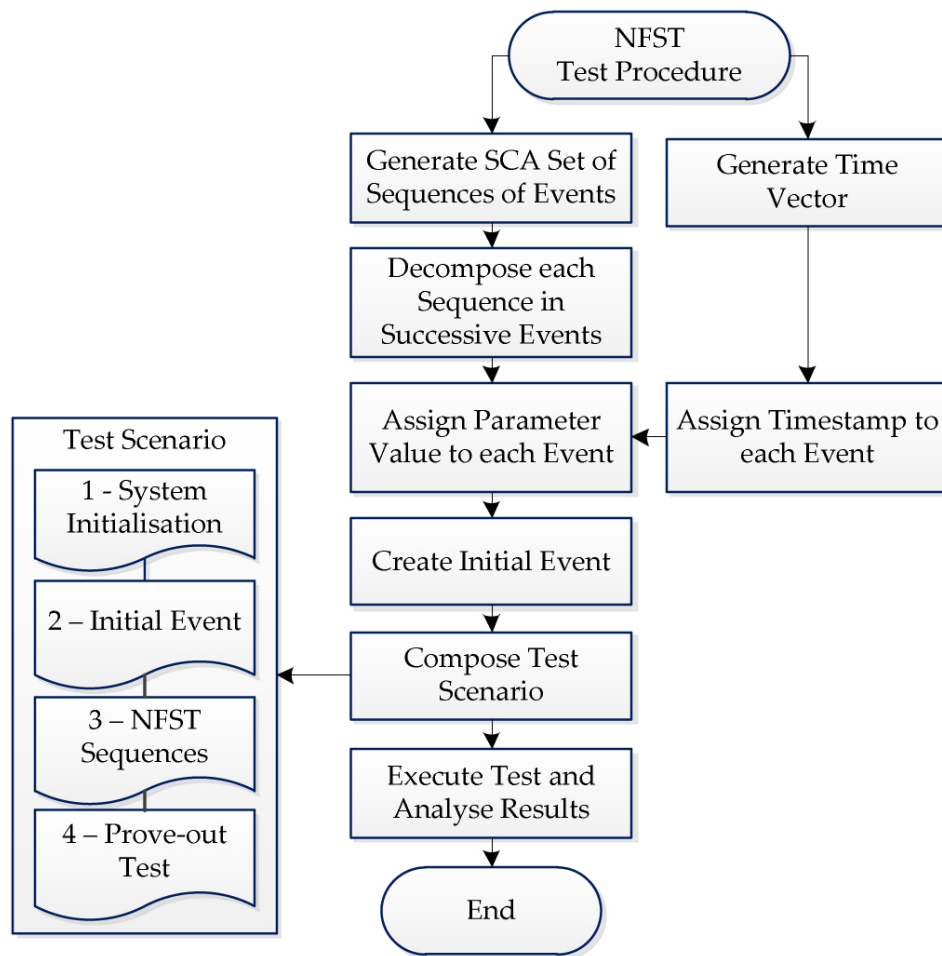


Figure 2.19 NFST approach (Becci et al, 2013)

The approach, while having similarities with TPT in generating test sequences from combinations of elements, is different in introduction of pseudo randomness in timing between events and in being more concerned with the ability of a system to recover the correct functionality rather than monitoring behaviour throughout the test sequence. The latter point is important when generating a large number of test cases and looking for unspecified robustness failures as there is no readily identifiable test oracle for this.

2.11 Formal methods

The use of formal methods, which mathematically prove the correctness of algorithms, models or software are one of the techniques recommended in functional safety standards such as IEC61508 and ISO26262 for very high safety integrity level software. An opportunity in the context of this thesis would be the use of formal methods to check robustness relevant properties in any system, rather than restricting their use to safety relevant properties in a safety critical system.

Woodcock, Larsen, Bicarregui and Fitzgerald (2009) reviewed the state of the art in the industrial application of formal methods. They note that, while formal methods can be used at any stage in a project, they are increasingly used at earlier stages of specification and design. They surveyed 62 industrial projects known to have employed formal methods. The projects had a range of start dates, a small number dating back to 1980's, but the majority being started between 2000 and 2008. The projects covered a range of application domains, of which transport was the individual domain with most projects (16) but it was not specified what aspect of transportation this covered. Examples of projects are given from aerospace and rail domains but none from automotive. Figure 2.20 shows the results of the survey in terms of the specific techniques that were used in projects illustrating the bias towards the formalization of the up-front activities, especially of specification and modelling.

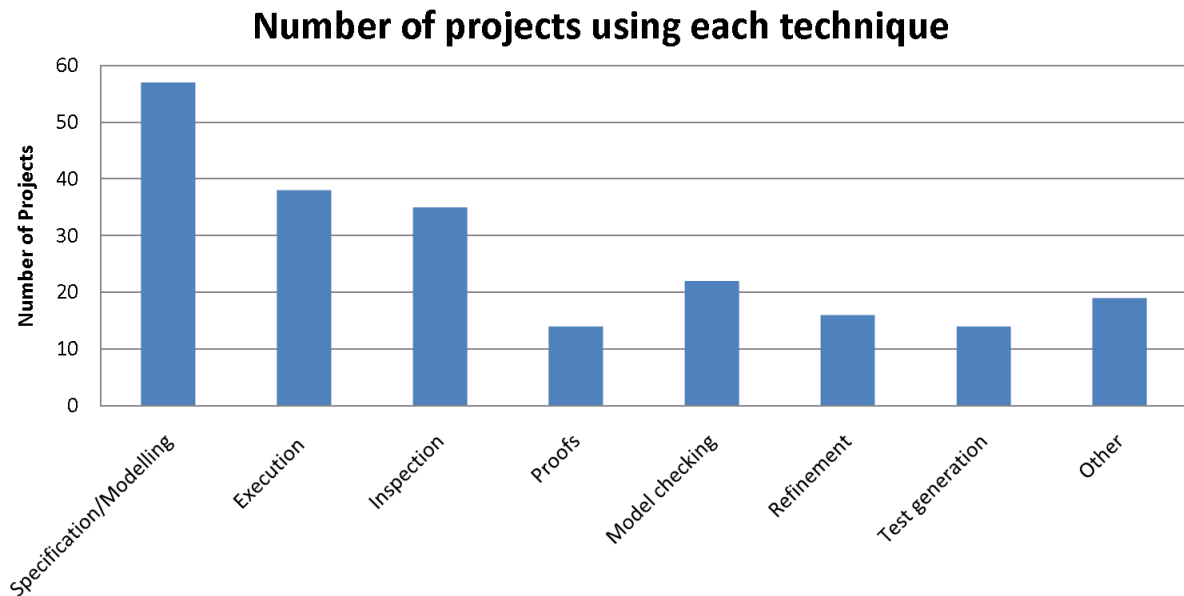


Figure 2.20 Techniques the survey of formal methods application by Woodcock et al. 2009

Respondents' experiences were generally favourable in terms of reductions in time and cost but they found it difficult to judge the actual savings from a reduction in issues later in projects from the increased effort to deploy formal methods in the early phases. Only half of the respondents reported the cost consequences and a key finding is there is still a lack of evidence for the cost benefit of such methods.

Respondents were highly favourable in terms of increased quality with 92% citing an improvement compared to other techniques. Overall in 95% of cases it was judged that the use of formal techniques were successful. While these early adopters are likely to be advocates of the techniques it is indicative of wider potential.

Challenges were noted in areas of: the use of tools by those who are not formal methods specialists ("mere mortals"), integration within existing toolsets, lack of automation and the speed of response. The authors build on this by commenting on the potential for lightweight formal methods focused on tool supported automated verification of particular properties of sub-systems. Specifically they note that "*the increasing capability of automated*

formal analysis makes it possible to have an impact on larger-scale developments, not necessarily critical, that are driven by the need to reduce time to market, defect rates or costs of testing and maintenance.” This aligns to the contention within this thesis that design methods used for safety critical systems can be viably used to increase the robustness of system that are not safety critical.

Given this potential for the use of formal methods in other domains what evidence exists for the use within the automotive domain? The use of formal methods within automotive has previously been reported as being limited in a 2008 survey by Rao, McMurran and Jones (2009). The reasons for this were, as previously noted by Rushby (2002), that formal methods still require a high degree of effort both human and machine (“*brute force*”) and focus on the steady state operation not conditions such as initialisation.

However more recent research is attempting to overcome these shortcomings through mechanising formal verification of models. Both the model frameworks discussed previously, EAST ADL and Compass have been and continue to be a focus for formal analysis using model checkers.

Qureshi, Chen, Persson, and Törngren (2011) describe work to transform execution timing constraints described in East-ADL models to be used by the UPPAAL tool, a formal model checking tool which can check for reachability, safety and liveness properties of real-time systems. They highlight this is as ongoing work with outstanding research challenges.

Chen, Feng, Qureshi, Lönn and Hagl, F. (2013) report on transforming East-ADL behaviour models to permit formal analysis using a model-checker called SPIN. While East-ADL does not model behaviours it does allow a declaration of triggering policies e.g. time triggered event triggered, timing and synchronization constraints. Additionally a new East-ADL package has been recently added referred to as *behaviour description annex* which can provide links to related behavioural models for example in Matlab Simulink. Chen et al

demonstrate how various behaviour constraints can be modelled in East ADL and a subset (not all) transformed into a format which can be used by the SPIN model checking tool and used to check the related behavioural model referenced in the behaviour description annex. The case study doesn't make clear the degree of success achieved however it is worth noting that SPIN was successfully used by NASA (2011) in their investigation of alleged unintended acceleration events in Toyota vehicles.

The COMPASS project (Oliveira, Sampaio, Antonino, Ramos, Cavalcanti and Woodcock, 2012) aims to translate its modelling language (CML), derived from SysML description of a system of systems, into a format amenable to formal verification in this case using the Circus tool which uses CSP (Communicating Sequential Processes) semantic models. Again this work is at an early stage.

Rajeev, Mohalik and Ramesh (2013) report on the formal verification of Simulink/Stateflow models using Matlab Simulink Design Verifier. They discuss the types of requirements that can be verified, pointing out that these must firstly be capable of being formally, i.e. precisely, stated and must be included in the model i.e. cannot verify system interactions not included in the model, and they propose a categorization for these. They propose a methodology based on specification templates and report successful results on case studies based on individual controllers rather than distributed systems.

Matsubara, Sakurai, Narisawa, Enshoiwa, Yamane and Yamanaka (2013) present an approach using a technique called "program slicing" to limit verification targets in order to avoid failure to complete verification through state explosion. In this approach hardware and software are modelled. The software is a limited subset of the source code related to variables of interest which is converted to software model. The hardware, parts of software and controlled equipment are modelled more abstractly as an external environmental model. These are combined into a verification model which can be checked using a model checker

such as SPIN. They have applied this approach to find logic and timing related malfunctions to several automotive control systems confirming the usefulness of the method. They also highlight usefulness of this technique in regression testing after revisions to source code.

From the above examples it can be seen that there is potential in formal methods in automotive electronic applications and that, although the tools and their application may not yet be sufficiently mature and lightweight beyond that of high integrity systems, a framework for robust design should incorporate them as an approach.

2.12 Discussion

From the literature robustness can be defined as dependability of a system with particular respect to external faults. Taking this definition of robustness as a specialised case of dependability and considering the base definitions of dependability raises the following discussion points.

The first definition of dependability given by Avižienis et al (2004) is one of it being the “*ability to avoid service failures that are more frequent and more severe than is acceptable*”. This highlights a need to specify what service failures are acceptable as a result of external failures and how often they can occur in order to have a measure of what is acceptably robust. While for safety critical functionality this may be straightforward for other functionality it may not be obvious, particularly without prior knowledge of what external failures could occur. This may require the specification of; degraded modes of operation, events that must not take place or availability targets. A related question for non-safety critical automotive systems is what are the key dependability characteristics and which are most impacted by external faults?

The second definition of dependability given by Avižienis et al (2004) is “*the ability to deliver service that can justifiably be trusted*” which raises the issue of what are the methods to gain assurance that the system can be trusted to be robust?

Complex systems are characterised by having interdependencies which make them inherently non-linear and difficult to model or abstract and exhibit emergent behaviour not recognised during the original design. Automotive Electronics are complex embedded Systems of Systems which are interconnected by design, have a high degree of “Dynamic Complexity” due to the temporal nature of dynamic interactions. There is evidence of the scale of issues of automotive electronics but not of the specific causes . The current automotive approach to cope with complexity and rate of change are standards for open systems, sharing and reuse of software and model-based development.

Robust design techniques, e.g. Design for Six Sigma, are mechanically focused and have an emphasis on a “Right First Time” philosophy and an expectation that failure modes are known in advance that fundamentally misses the issue of emergence. This contrasts with the learning from the field of software development, particularly agile methods, on emergent issues which is to develop iteratively and incrementally to allow continuous integration and test. Hence any robust design process framework for complex systems needs to build in an iterative approach.

While there are explicit design methods for achieving mechanical robustness, for software-based systems the main concern appears to be reliability and prevention of fault conditions in the first case. This may be because the major research efforts and standards focus on high dependability systems but this tends to lead to solutions such as guaranteeing high reliability or redundancy which may not be appropriate for non-safety critical systems which nonetheless need to be robust to meet customer expectations. This primacy of reliability over robustness in the literature may reflect research being driven by the demands

of high integrity systems such as for aerospace or safety critical automotive applications rather than the increasing demand of mass complexity of the majority of automotive applications and consumer goods.

However there are many good design principles which do not entail additional cost but are not fully recognised in the automotive field. For example there seems to be little or no published work considering the application of concepts of self-stabilization within the automotive domain. It may be that the principles of self-stability are implicit in automotive design and often fail-safe is achieved by reverting back to sufficient level of mechanical control e.g. in braking and steering systems with a reboot of the microprocessor to re-invoke the additional electronically controlled functionality. However this may benefit from some explicit consideration. Certainly it suggests a need for systematic knowledge capture and dissemination which while being beyond the scope of this work should be included as an element in a framework for design for robustness.

There are other lessons to learn to from safety critical approaches such as the need to do upfront risk analysis to determine where design effort to address systematic risks needs to be. Safety cases are a specific tool which it is proposed to investigate whether they can be adapted to designing for robustness.

The use of formal methods has significant potential within automotive and is the subject of on-going pilot and research projects focusing initially on high integrity applications. This should pave the way for lightweight methods and tools which then can be used for robustness analysis of all systems. To enable this requires design artefacts for formal analysis, models in particular, which encompass robustness critical attributes and the knowledge of what are the specific properties that need to be checked to prove robustness. While the development of the formal methods tools and techniques is outside the scope of this thesis, the framework for robust design should recognise the potential to apply such

methods to robustness issues and provide enabling capabilities for this in terms of models and the understanding of the specific properties that need to be checked to prove robustness.

A significant question arising from the literature is: can complex systems of systems be usefully modeled at a level of abstraction that allows robustness properties to be examined? There is extensive work ongoing on modelling complex systems which can provide a framework but whether modelling right things in terms of robustness is debatable in the absence of a clear understanding of robustness issues.

This highlights the importance of domain knowledge in complex systems. Domain knowledge has been identified as critical in the literature for: developing design guidelines, recognising critical robustness factors, modelling and abstraction and identification of areas of focus for robustness testing. For example the final statement of Driscoll et al. (2003) on practical approaches to Byzantine failures is “*Anyone designing a system with high dependability requirements must thoroughly understand these failures and how to combat them.*” However there is a lack of domain knowledge for automotive electronics robustness issues in published literature, this may be due in part to an unwillingness to put such information to the public domain and in part due to a lack of specific systematic robustness focused studies. Hence to meet the objectives of this thesis a systematic study of robustness issues is required to build domain knowledge.

Test methods addressing robustness is an area where there is a growing body of work which should be included in a design for robustness framework but not subject to specific development within the scope of this thesis.

2.13 Conclusions

From this literature review the key opportunities for contributions to knowledge in the field of robust design of automotive electronics are as follows:

1. Development of domain knowledge of impact in terms of prevalence, dependability characteristics impacted and the causes of robustness related failures in the area of automotive electronics.
2. Development of a design for robustness framework for complex automotive electronic systems. This should leverage best practices identified during the literature review but should include the following two areas identified for further new work.
3. Adaptation of safety cases to robustness to enable structured arguments for assuring the robustness of a system.
4. Development of an approach to robustness modelling based on understanding what are important factors to model pertaining to the robustness of automotive electronics.

The remainder of this thesis is based around achieving these contributions. The next chapter will focus on the development of domain knowledge of robustness failures in the field of automotive electronics.

Chapter 3 - Case Study Review of Automotive Electronics Robustness Issues

3.1 Introduction

In Chapter 2 a literature search was conducted to understand the nature of complex systems and potential methods which could be employed to improve their robustness, i.e. their ability to provide specified acceptable levels of service during and after external faults. It also highlighted a lack of published literature relating to domain knowledge of the impact and the causes of robustness related failures in the area of automotive electronics. In this chapter a case study of a significant number of automotive electronic issues is undertaken to firstly confirm that robustness is indeed an issue in automotive electronic systems, to consider what dependability characteristics are effected and to identify and analyse the types of faults that lead to root causes of robustness failures. This research is intended to be informative for the subsequent work developing a design for robustness framework and supporting methods to ensure it is well targeted.

This chapter presents the methodology, results and conclusions from this case study review including discussion of the validity and reliability of the results. The details of specific issues are not divulged but the findings are most significant at an aggregated level in terms of direction of the work to address the wider set of issues rather than a particular instance.

3.2 Case Study Methodology

3.2.1 Case Study Questions

The specific questions which the analysis would seek to answer to ensure the subsequent work developing a design for robustness framework and supporting methods is well targeted are as follows.

- What is the prevalence of robustness issues in automotive electronics? A fundamental question relates to whether robustness related issues are really a major problem. Although this was an assumption based on experience of the field there was not substantiating empirical analysis available to indicate the actual proportion of problems with robustness related issues.
- What are the impacts of robustness failures, specifically what dependability attributes are affected?
- What are the types of faults that lead to robustness failures at a generic and specific level? Are these faults related to hardware or software or both? What is nature of the activation mechanisms which cause these faults to lead to service failures.
- What are significant robustness related parameters? For those issues that were robustness related then the analysis should identify what are the contributory factors resulting in faults.
- What were the subsequent actions resulting from the issues e.g. changes to design, design process, specifications or validation methods? Understanding for a number of specific issues this may give an insight into where weakness may exist at a more general level.

3.2.2 Selection of Cases to Review

The criteria for selection of the case studies were as follows. The case studies must be examples of service failures in automotive electronic systems as these are the specific focus of this work. The system exhibiting the service failures must be electronic involving some form of programmable hardware, not purely electrical e.g. failures in wiring harnesses. The service failure should have occurred in the operational use phase, i.e. when the vehicle has completed its development program, to ensure that faults are ones which had not been detected by current methods and were of particular concern as the customers could experience service failures. The faults should be representative of all faults within these criteria to be sure that conclusions on the prevalence of robustness faults are valid. The case study issues need to have been analyzed in depth and the true root causes identified and documented in detail. The case studies should be sufficient in number to support the validity of conclusions.

Two sources of information which fitted these criteria were identified to be used for the study from a single automotive manufacturer. The first was known as “Prevent Action Closure Papers” (PAC). These were the documents for significant issues which had led to either; service actions to repair vehicles in the field or “gateholds” where new vehicles are quarantined for repairs before shipping. These were issues that were deemed of a level of concern that they would be formally reviewed by senior management to ensure that not only was the particular issue fixed but also that it would not re-occur. The documentation contained a description of the issue and its impact, the design or process related root causes and the actions taken to prevent the recurrence of the specific concern and similar concerns in the future. 25 PAC papers were provided for review based on being most recent examples and discussed with the Departmental Quality Manager. Of these 18 fitted the selection

criteria, the reason for rejection of 7 issues being that the example was a purely electrical, as opposed to electronic, issue.

The second source of data was 6 Sigma Project reports. 6 Sigma is a data-driven problem solving technique that seeks to systematically identify and remove sources of defects and variations that lead to quality issues, often by the application of statistical methods (Pande, Neuman and Roland, 2001). The data provided conformed to the steps in a “DMAIC” process, named after the phases of Define, Measure, Analyze, Improve and Control. In the Define phase the key issue which is to be addressed is identified in detail and in particular what the customer effect is. In the Measure phase data is gathered relating to the problem including assessment of process capability. The Analyze phase seeks to get to the true root cause of the problem followed by the Improve phase where the effectiveness of potential solutions is measured and optimized. The Control phase ensures new solutions are introduced into production rigorously with adequate methods to ensure their effectiveness is monitored. There is a final Replicate phase where measures to prevent the same problem happening on future products or on other product lines are put in place, evidenced and signed off. 26 completed 6-Sigma projects were reviewed including discussions with the relevant 6 Sigma “Master Black Belt” who is the champion and coach for 6-Sigma within the particular department. Of these 18 were selected as meeting the necessary criteria.

3.2.3 Analysis Method

To conduct the analysis of the case studies against the identified questions as objectively as possible, criteria for the categorization were developed, based where possible on existing definitions.

3.2.3.1 Prevalence of robustness issues

Based on the definitions identified in the literature review a robustness failure pathology to be used within the analysis was developed which is shown in Figure 3.1.

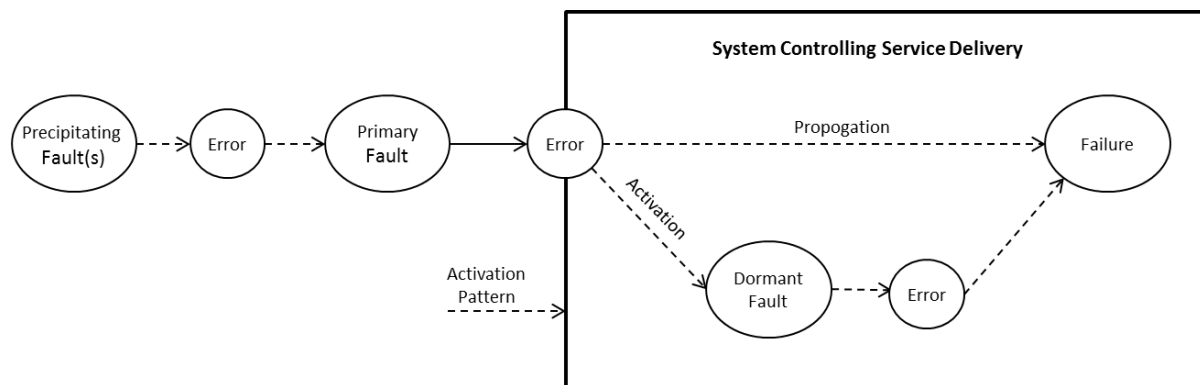


Figure 3.1 Robustness failure pathology used within case study analysis

This pathology is in line with the definition of robustness failures being those which occur as a result of an external fault. Failures in which no fault occurred external to the system controlling service delivery are not considered robustness failures. The boundary of the system is considered to be the programmable device with responsibility for controlling delivery of the failed service and any dedicated sensors and actuators. In cases where the service is delivered through a human machine interface device, such as a display screen, the programmable device with the responsibility for the service is that providing the source not the device controlling the interface. The primary fault is the direct cause of error that is propagated into the system causing the service delivery failure either directly or indirectly through activation of a dormant fault within the system. The primary fault may in turn have been caused due to errors as a result of other precipitating faults. An activation pattern of inputs may be required to create conditions for the error at the system boundary to lead to the service failure.

3.2.3.2 Impact of robustness issues

The primary impact of robustness of the faults on dependability attributes is assessed by selecting one only of the following:

- Availability: readiness for correct service (service not provided to user)
- Reliability: continuity of correct service (service interrupted or degraded)
- Safety: absence of catastrophic consequences on the user(s) and the environment
- Integrity: absence of improper system alterations
- Maintainability: ability to undergo modifications, and repairs

In cases where there are more than one dependability attribute impacted then the most severe, in terms of effect to the end user, is selected.

3.2.3.3 Fault categorisation

The fault categorization scheme is based on the elementary fault classes defined by Avižienis et al (2004) illustrated in Figure 3.2.

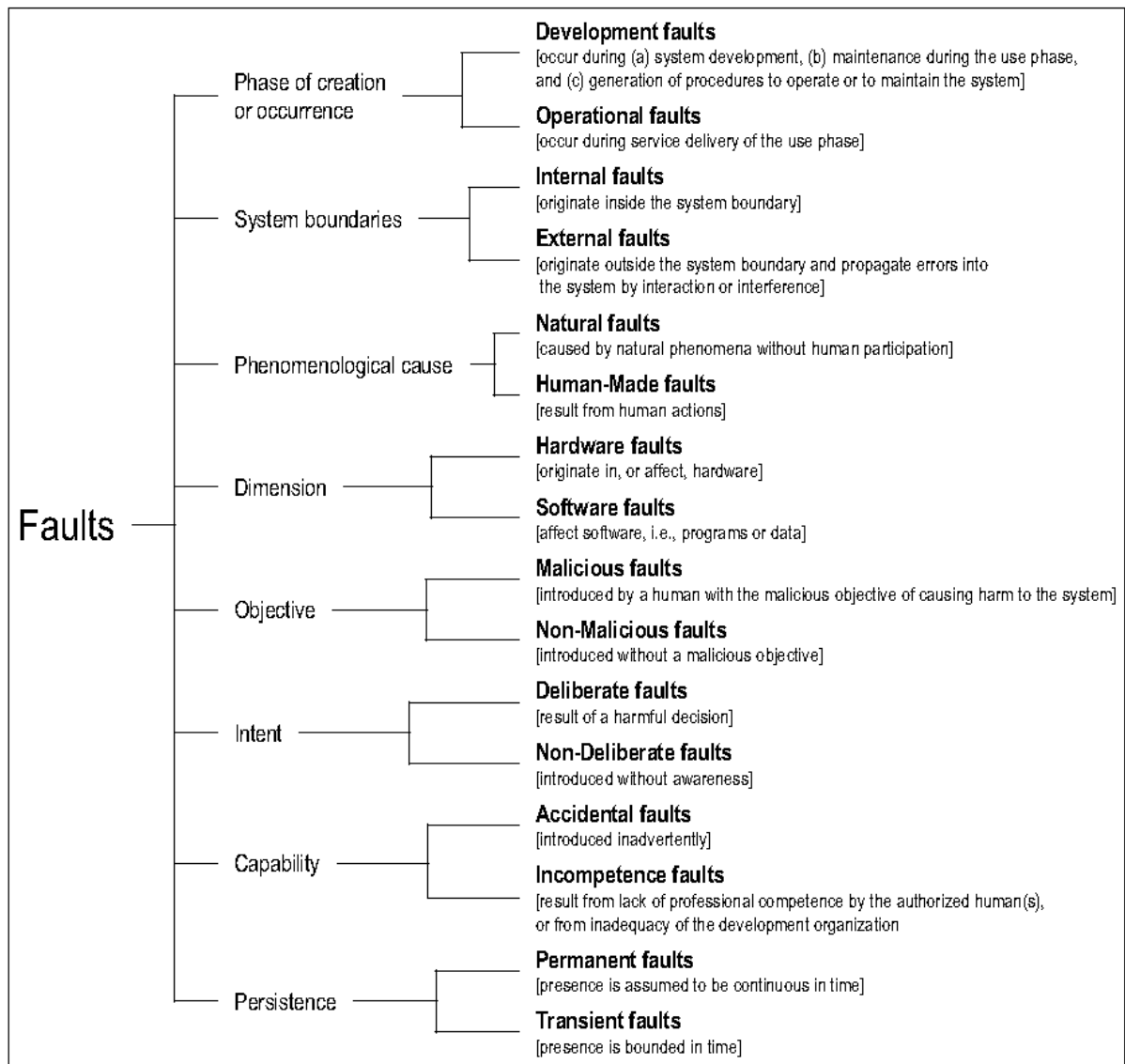


Figure 3.2 Elementary fault classes defined by Avižienis et al (2004)

Further classification and tailoring of this scheme was found to be necessary. Avižienis et al (2004) is ambiguous for the first fault class in that it is termed as the “phase of creation or occurrence”. In the case of elusive faults they may be created during development but not seen to occur until operation. To remove this ambiguity the case study considers purely the phase in which the fault is created. This requires an element of judgment as to whether it is reasonable that the issue could have been prevented if the right tools and techniques had been available to predict the fault and add preventative measures in the development of the system

or whether the fault could only be apparent in the operational use phase of the system. The original work appears to give little consideration to mass production of systems whereby manufacturing is an important area where faults could be created so this has been added a distinct category.

The system boundary was defined as the boundary of the system exhibiting the service failure. By the definition all faults which originate outside the system boundary and propagate errors into the system by interaction or interference are robustness faults.

At this stage it is necessary to augment the taxonomy to distinguish between the primary fault at the system boundary and the precipitating faults which may have led to this through fault propagation. For example in the case of a system which exhibits a service failure as a result of interaction with another system which has experienced electromagnetic interference, the primary fault is an interaction fault and the secondary fault is a physical fault.

The phenomenological causes of the error at the system boundary are either natural faults which are physical hardware faults resulting from natural phenomena, e.g. extreme temperatures, without human participation or are else deemed to be human-made. The dimension parameter notes whether this error effects hardware or software of the system controlling the service delivery.

From the data provided it is not possible to reliably determine the objective, intent and capability of human made failures so these factors are omitted from the analysis.

Persistence is defined as the persistence of the fault as opposed to the service failure, noting that a transient fault can lead to permanent service failure and vice versa. At this stage the activation pattern is not accounted for i.e. if the error caused by the fault is permanently at the boundary of the system then it is deemed a permanent fault even if it requires a specific activation pattern to cause a service failure.

3.2.3.4 Phenomenological Fault Causes

As well as classifying by the elementary fault classes above, to answer the question of what are the phenomenological causes of robustness issues at a more specific level a bottom up classification scheme will be used in parallel. In this approach the adjudged failure causes will be used to create specific classes of fault causes with each fault cause being allocated to an existing class, either directly or through generalisation of the class, or creating a new class. This is done for both the cause of the primary external fault and for the precipitating faults.

3.2.3.5 Activation pattern

To gain insight into the activation patterns for the robustness faults to manifest themselves as failures for each issue the number of factors and the specific factors are noted. The factors can be considered as the independent variables which would need to be controlled in an experiment to make the error manifested itself as a service failure.

3.2.3.6 Resulting Actions

A bottom up classification scheme will be used to create specific classes of resulting actions which have been taken as a result of the issues. In the case of design changes specific classes for software, hardware or both will be used to categorise resulting actions.

3.2.3.7 Earliest Potential Detection Point

A key question in looking to implement effective solutions to robustness issues is; what was the earliest point at which a particular issue could have been detected. This analysis requires a degree of subjective interpretation in many cases as to what was considered

reasonable, as it could be argued that if you had perfect foresight and complete specifications then most issues could be picked up in Failure Modes and Effects Analyses (FMEAs) at the specification stage. The detection points selected were based on stages from a V-Model view, shown in Figure 3.3, of the system development lifecycle running from requirements, specification and modelling on the left-hand side and unit, system and vehicle test at the right-hand side.

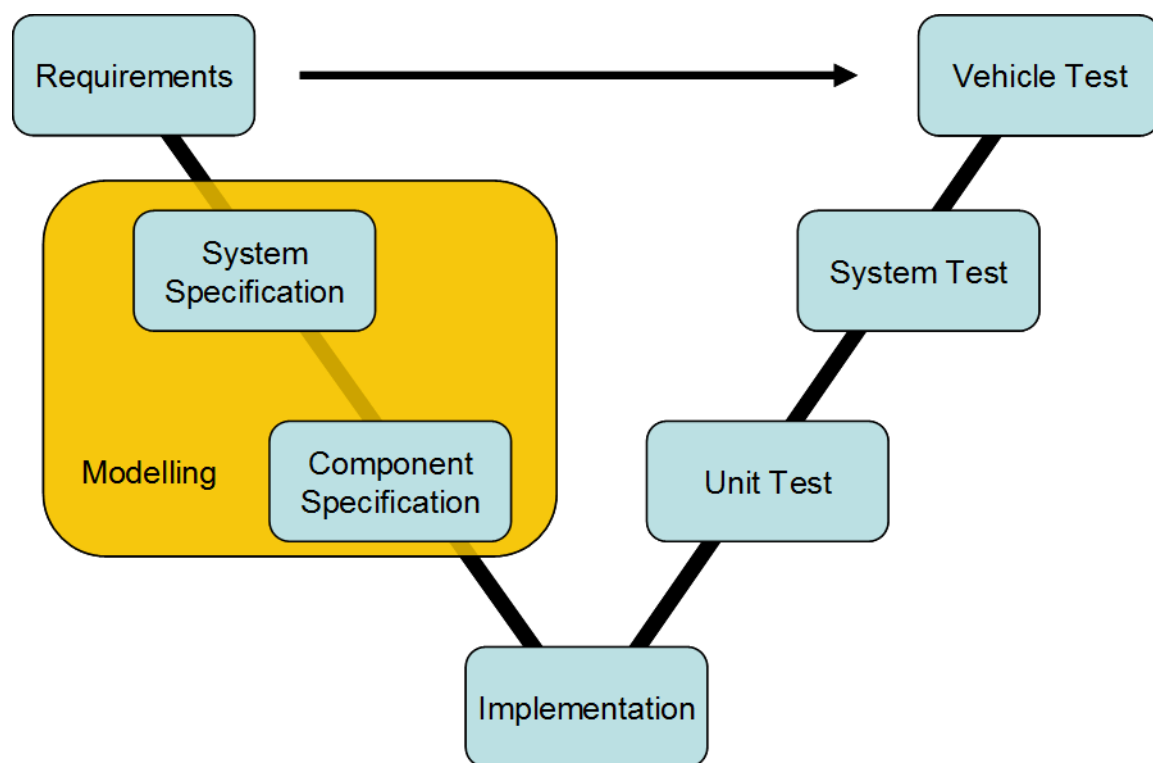


Figure 3.3. Systems V-Model in automotive application

Additional categories were added for diagnostic validation and manufacturing test at a component and vehicle level.

The analysis was carried out by the author reviewing each test case and for each issue completing a categorisation on a spreadsheet based around the definitions above to give comparable results for from which conclusions could be drawn.

3.3 Case Study Results and Analysis

3.3.1 Overall Occurrence of Robustness Issues

From the 34 selected case studies of automotive electronic failures a total of 43 issues were identified for analysis, as some case studies covered multiple issues. These were analysed according to the robustness failure pathology and fault categorization identified in the methodology. 26 of the 43 (60%) of all issues were categorized as robustness related, i.e. due to faults external to the system controlling the service delivery.

3.3.2 Impact of robustness issues

The primary impact for the robustness faults only in the case studies on dependability attributes is shown in Table 3.1.

Primary Impact of Robustness Faults		
Availability: readiness for correct service (service not provided to user)	10	38%
Reliability: continuity of correct service (service interrupted or degraded)	15	58%
Safety: absence of catastrophic consequences on the user(s) and the environment	0	0%
Integrity: absence of improper system alterations	0	0%
Maintainability: ability to undergo modifications, and repairs	1	4%

Table 3.1 Primary impact of robustness faults in case studies on dependability attributes

From this it can be seen that the primary impact is that the service provided to the user is degraded in 58% of cases and its availability is reduced in 38% of the cases. In no cases were there any safety impacts but it is possible that if such cases existed they were not disclosed. That there are no integrity issues is not unexpected as the vehicle systems, unlike IT systems, are not “open” and require specialist tools to access and make changes.

3.3.3 Fault Categorization

The categorisation of fault according to the defined elementary fault classes is shown in Table 3.2.

Fault Classification Scheme						Results	
Creation Phase	Fault Origin	Cause	Dimension	Persistence	Type	No.	%
Development	Internal	Human-made	Software	Permanent	1	2	5%
				Transient	2	2	5%
			Hardware	Permanent	3	5	12%
				Transient	4	5	12%
	External	Human-made	Software	Permanent	5	1	2%
				Transient	6	19	44%
			Hardware	Permanent	7	0	0%
				Transient	8	3	7%
Manufacture	Internal	Human-made	Software	Permanent	9	2	5%
				Transient	10		
			Hardware	Permanent	11		
				Transient	12		
		Natural	Hardware	Permanent	13	1	2%
				Transient	14		
	External	Human-made	Software	Permanent	15	2	5%
				Transient	16		
			Hardware	Permanent	17		
				Transient	18		
		Natural	Hardware	Permanent	19		
				Transient	20		
Operational	Internal	Human-made	Software	Permanent	21		
				Transient	22		
			Hardware	Permanent	23		
				Transient	24		
		Natural	Hardware	Permanent	25		
				Transient	26		
	External	Human-made	Software	Permanent	27		
				Transient	28		
			Hardware	Permanent	29	1	2%
				Transient	30		
Natural	Hardware	Permanent	31				
		Transient	32				
Total						43	100%

60% Robustness Issues

Table 3.2 Categorization of case study faults according to the elementary fault classes

Overall it can be seen that the vast majority (86%) of the issues were adjudged to have been created in the development phase, with a smaller amount (12%) during manufacturing and only a single issue adjudged to have been created in the operational phase.

The non-robustness related issues (40% of total) related predominantly to component design (fault Types 1-4 - 34%) and manufacturing issues (Types 9 & 13 – 7%) internal to the

system. Some of these were elusive faults, such as failure of micro-processor initialisation at very low incidence, but were internal to the system, rather than the result of any error propagated into the system and so not robustness issues.

Of the robustness issues the most prevalent fault type is Type 6, accounting for 44% of all issues and 73% of the robustness issues. These are transient faults, created in the development stage, originating outside the system and affecting software. These were typically types of communication faults. The next most common robustness issue is Type 8 but with a relatively low level of 7% of all issues and 12% of the robustness issues. Type 8 issues are caused by transient faults, created in the development stage, originating outside the system, and affect hardware. In these cases the error at the interface is physical in nature, e.g. electrical or electro-magnetic signal, but not naturally occurring.

3.3.4 Robustness Issues – Phenomenological Fault Causes

For the robustness issues the phenomenological fault causes for both primary external fault and for the precipitating faults were classified. Figure 3.4 shows the Primary phenomenological causes of robustness issues that are the errors seen at the boundary of the system controlling service delivery.

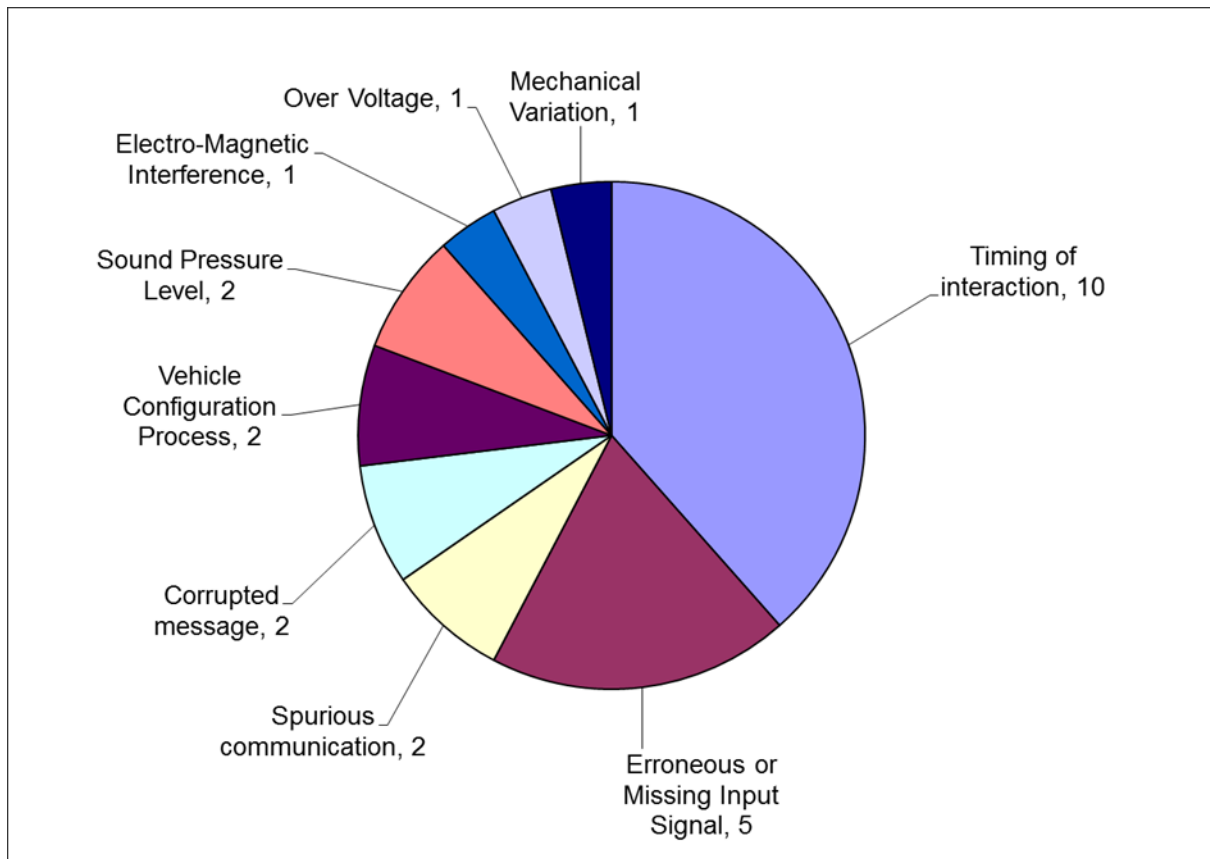


Figure 3.4 Primary phenomenological causes of robustness issues

The causes are dominated by those leading to communication errors (73% of total), the most significant of which is the timing of interaction. This causal class includes race conditions, early and late interactions that lead to anomalous results and service failures. The next most significant class is erroneous or missing input signals whereby an external system either provides a wrong value of input or failed to provide an input leading to failure in service delivery by the receiving system. The final classes relating communication errors are spurious communications or corrupted messages where the receiving system receives invalid inputs.

Non-communication related causes include errors in external configuration files which set the behaviour of the system, and physical phenomenon where variation has exceeded specified or expected levels.

Figure 3.5 shows the precipitating phenomenological causes of robustness issues which were the indirect causes leading to the external faults. Note that some issues had more than 1 precipitating cause and some did not have any.

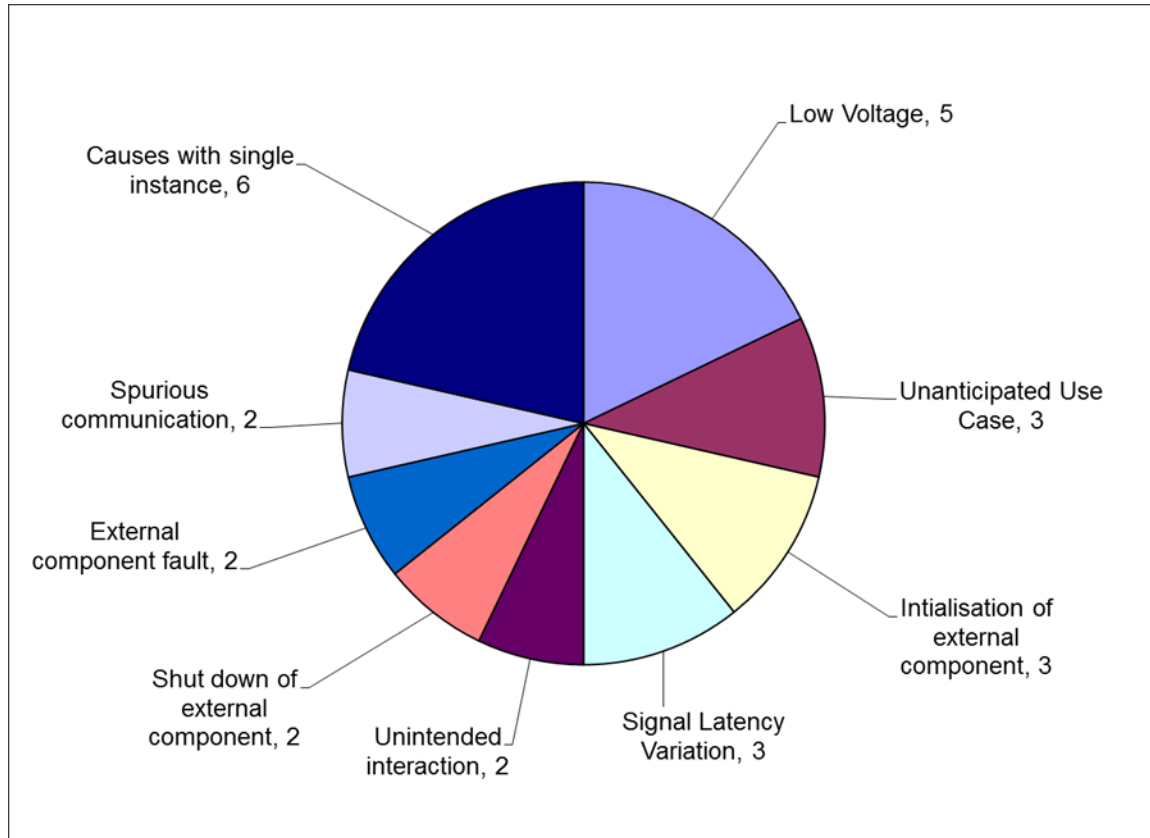


Figure 3.5 Precipitating phenomenological causes of robustness issues

The largest single issue relates to low supply voltage transients, which predominantly occur when the engine is being started. These cause some ECUs to shut down and re-initialise while others are still running due to variations in the supply voltages and operating voltages of individual ECUs. This in turn leads to communication errors which were seen as direct causes of robustness issues. The high number of robustness issues due to the initialization and shut down behaviour of the electrical systems corroborates findings reported by Huang et al. (2009) which resulted in new verification methods for transient low

voltage events. While new verification methods are a partial solution a key question is; can such issues be designed out in the first place?

There were 3 cases where there were unanticipated use cases. These related to user interactions with new systems for which there was not an established pattern of user behaviour and so were not adequately considered in the design of communicating systems. This leads to missing or wrong communications to the service delivery system.

There were 3 cases where variations in signal latency were the cause of interaction timing errors seen by the service delivery system. Other cases where more than one instance was noted were faults in components and spurious communications external to the system leading to erroneous inputs into the system.

3.3.5 Activation Patterns

Table 3.3 shows the results of the analysis of the number of factors in the activation pattern of the robustness issues in the case studies.

Number of Factors		
Permanent	2	8%
Single Factor	10	38%
2 Factors	8	31%
3 Factors	6	23%
>3 Factors	0	0%

Table 3.3 Number of factors in activation pattern of robustness issues in case studies

There were 2 robustness issues with permanent activation conditions – purely the presence of the external fault would lead to a service failure. In these cases the faults

themselves were of very low incidence. There were a large proportion of the robustness issues where only a single factor was involved in the activation pattern. Further analysis showed that in these cases either the activation factor was an analogue value that had to be very precisely controlled with respect to timing or level or it was combined with an elusive fault. Most issues needed 2 or 3 activation factors which contributed to the difficulty in detecting them. Table 3.4 shows the specific factors in activation patterns of robustness issues in case studies.

Specific Factors		
Particular User Operation	13	50%
Low Voltage Transient	7	27%
Vehicle Power Mode behaviour	6	23%
Service Mode	2	8%
Operation of particular system function	2	8%
Specific Configuration	1	4%
High Voltage Transient	1	4%
Ground Line Interruption	1	4%
Disconnection of 12V supply to vehicle	1	4%
EMC	1	4%

Table 3.4 Specific factors in activation pattern of robustness issues in case studies

Particular actions by the user, e.g. invoking of specific functions, was the most common factor being necessary for 50% of the issues. Other more significant factors were

low voltage transients, which cause shut-down and initialisation behaviours, and particular vehicle powermode states where vulnerabilities only exist in specific powermodes.

3.3.6 Resulting Actions

For failures related to electronics a key question was whether the failures were related to hardware or software. A related issue is whether the resulting modifications were related to hardware or to software, a working hypothesis being that software changes would be seen as the most expedient to implement. The results of the analysis are shown in Figure 3.6 below.

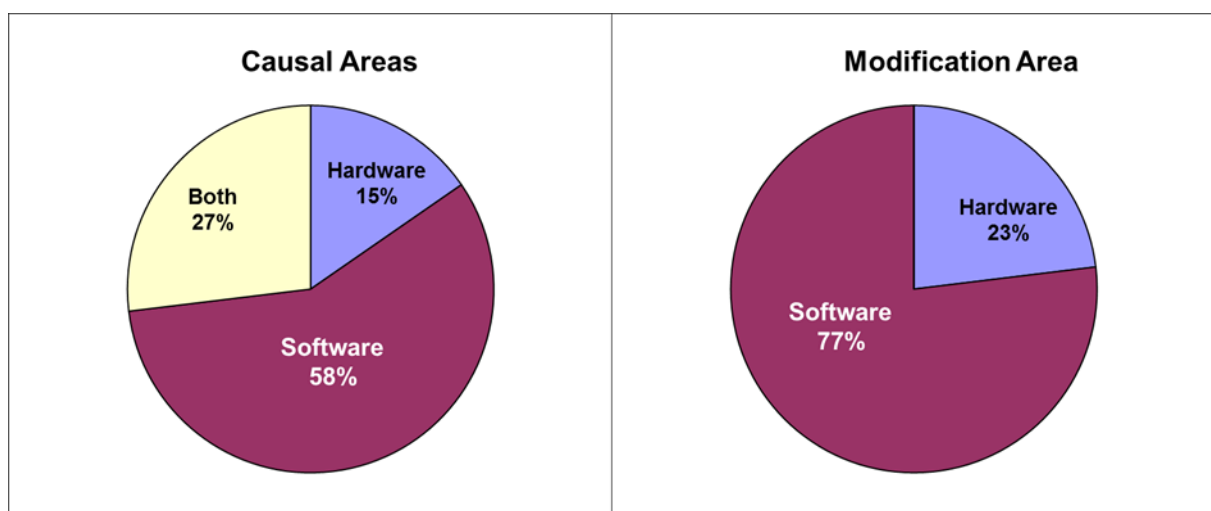


Figure 3.6. Electronics robustness issues - hardware and software causal areas and modified areas

From these results it can be seen that the majority of electronics issues due to robustness failures were completely software related and in total software was a contributory factor in 85% of failures with only 15% being purely hardware related failures. As expected, the modifications resulting from electronics issues were predominantly software and in a

number of cases a software change was used to protect against hardware issues through some form of defensive programming.

To further help the understanding of how the design and development process can be enhanced to improve product robustness the actions which were taken as a result of the robustness issues were analyzed. The results are shown in Table 3.5. Note that the table shows the percentage of cases in which a particular action was taken and that in many cases there were multiple actions.

Resulting Actions		
Design Change	23	88%
Manufacturing Process Change	3	12%
Design Spec Updated	18	69%
Verification Method Updated	16	62%
FMEA Updated	11	42%
Design Review Updated	7	27%

Table 3.5 Resulting actions to robustness issues

Unsurprisingly in the vast majority (88%) of cases the design had to be changed – the exception being manufacturing process changes (12%). While these represent a number of specific responses, the generic design documentation was also updated in 69% of the cases studied. A question is why it was not always updated in all cases where design changes were made and it is suspected that it was in many more cases but not noted in the documentation.

The verification method was also noted as being updated in 62% of cases mainly to extend the coverage, duration and conditions of the testing. This highlights one of the challenges of testing for robustness in that there are many potential parameters involved and that simply boundary or limit testing is not sufficient.

The FMEA was noted as being updated in 42% of cases indicating they are not effectively detecting robustness related failures. What is debatable is whether this is down to; the process itself, or the rigor with which is applied, or both. Robustness related faults which have a multi-factor activation pattern may often be too subtle and dependent on implementation factors to be picked up by a desktop analysis.

Finally the design review process was noted as being updated in 27% of cases. Design reviews are typically based on peer review of design artefacts with respect to best practice and “lessons learned” checklists of list particular areas of system design where previous issues have occurred.

3.3.7 Earliest Potential Detection Point?

The results of the analysis assessing where the earliest detection point could be for robustness issues are shown in Table 3.6.

Earliest Detection Point		
Requirements	0	0%
Specification	1	4%
Model	9	35%
Unit Test	7	27%
System Test e.g. HIL	3	12%
Vehicle Test	2	8%
Diagnostics Test	2	8%
Component Manf Test	0	0%
Vehicle EOL Manf Test	2	8%

Table 3.6. Estimated earliest detection points of robustness related failures studied

The first observation on the results is that the center of the V is potentially most critical in the detection of robustness related failures with 73% of the earliest detection points lying in modelling, unit test and system test. This is significant and promising result in that 77% of the issues could potentially be picked up before vehicle tests which are expensive, have limitations in testing robustness as they represent only a few instances of the systems and are restricted in their coverage of the overall test space. It is also interesting that for robustness related issues so few of them were thought to be reasonably detectable through reviews of requirements and specifications. This was as a result of the robustness related failures not always being present and manifest but only occurring when both a fault and its activation pattern are present. Hence to be able to detect such failures from a purely textual description is difficult - it is necessary to build a model, or a prototype to analyze and observe the effects of disturbances, variations and interactions. This is not to say that at a general level

requirements and specifications cannot be reviewed for areas where robustness may be a concern but in terms of finding specific issues there may be insufficient information to detect them at these stages.

Modelling appears to be a major opportunity in terms of the early detection of a significant number of failures, particularly those associated with communication failures. A key benefit of a model is the speed of iteration of the tests and the ability to introduce variation into the model as opposed to a physical part or software code. However with a model the key question is what attributes of the system under test have to be modeled and to what level of fidelity to get the desired results. From the case studies parameters related to robustness issues were identified earlier and these would then have to be incorporated in a model. These parameters are mainly not related to the primary function of the system and hence are not routinely modeled but are related to the ability of the system to function. Normal practice is to model the control logic or functionality of an ECU while making an implicit assumption that the underlying processing platform and data will be available as required. From the case studies it is clearly apparent that this is not always the case. This suggests a key opportunity for investigation is robustness models which represent the ability of a system to function and to interact with other parts of the overall system.

After modelling, unit testing and system testing are the other major areas of opportunity to detect failures early in the lifecycle. Unit and system testing are activities which are routinely done, yet had not picked up the failures studied. This therefore suggests that the test regimes are not conducting the right types of tests to pick up robustness issues. The question of what testing should be done is examined in the next section.

3.4 Threats to validity of study

Before considering conclusions, the limitations and threats to the validity of the case study need to be considered.

3.4.1 Selection of cases

The first area to consider is the selection of the case studies and whether they are representative of the wider set of automotive electronic issues. There was a conscious decision in the selection to consider only issues encountered in the production phase after the development lifecycle. Hence the issues are more likely to be the result of elusive faults, and should not be taken to be indicative of all faults during complete lifecycle, as it is to be expected that vast majority of faults will be found and rectified during development.

The case studies came from a single vehicle manufacturer so it needs to be considered whether they are more widely representative of other manufacturers. It is not possible to make a definitive conclusion without further datasets but from statements quoted in the literature review it is clear that issues with automotive electronics is a wider industry problem. Also the technology underlying vehicles of different manufacturers are similar, much of it based on shared standards e.g. CAN communications networks, and using the same pool of tier 1 suppliers, hence it seems likely issues will be similar. This still leaves the area of methods and tools as potential cause of differences in issues seen between manufacturers if a manufacturer has a particular technique for avoiding a type of issue.

A further concern is whether there is a sufficient quantity of issues? While overall the data is a reasonable sample size care needs to be taken over generalising from smaller incidences of faults in specific categories.

3.4.2 Selection of the categories

The selection of the categories was based where possible on existing definitions although these had to be augmented and clarified in areas to give more precise definitions to work from. However the specific issues were categorised on a bottom up basis which was based on the judgment of the person completing the studies and so may be less repeatable.

3.4.3 Categorisation Decisions

A limitation of the case study is that it was not possible to peer review and cross check the categorisation decisions as the base data has been given in confidence. This is partially mitigated by the work on defining the top down categories, on the fault pathology and through iterating the reviews to check self-consistency of decisions. Future work could include correlation studies by another researcher using the categories developed on new data sources.

3.5 Conclusions

3.5.1 Overview of Results

The case study was conducted to better understand the root causes of robustness related failures at both a technical and a systematic level to provide focus for further areas of work. The conclusions from the case study and their implications for an approach to design for robustness are given below.

The first question the case study set out to answer was what is the prevalence is of robustness issues in automotive electronics? The case studies showed the issues that are not detected through normal verification methods and reach customers are predominantly robustness related. Of the 43 faults analysed 26 (60%) were robustness related issues due to faults external to the system controlling service delivery, and hence it can be concluded that

robustness failures are a key issue to be addressed. This does not necessarily imply that this portion of all failures are robustness related as the case studies didn't include problems that were fixed during development or in-service issues that had more minor impacts. However it does suggest that current techniques are generally picking up simpler failure modes but not more complex robustness related failures which then manifest themselves as issues when the vehicle is in service causing customer dissatisfaction. It therefore verifies the need for work in identifying improved techniques to prevent robustness related failures.

It was found that the primary impacts robustness failures are on reliability, with 58% of issues studied giving interrupted or degraded service, and on availability to provide the correct service in 38% of the issues studied. None of the issues were safety-related but any such issues may have been pre-filtered.

The fault classification scheme identified the most common robustness fault, accounting for 44% of all issues and 73% of the robustness issues, as transient faults, created in the development stage, originating outside the system, affecting software. The primary cause of this type of fault was predominantly communication errors, the most significant of which is the timing of interactions (race conditions, early and late interactions) followed by erroneous or missing input signals, spurious communications or corrupted messages. In a significant number of cases the precipitating cause for such communication errors was shut-down and re-initialisation of external systems, often due to low voltage transients. These types of issues should be the focus of robustness methods to be developed.

Analysis of activation patterns of robustness failures concluded that while most were complex, involving 2 or 3 factors, a significant number (46%) only required a single factor or were permanent (i.e. if the fault occurred then so would the service delivery failure). Further analysis showed that in these cases either the activation factor had to be precisely controlled with respect to timing or level or it was combined with an elusive fault. Any robustness

methods need cater for a variety of activation scenarios including at least 3 factors. Particular factors which were highlighted were user operations, low voltage transients and vehicle power-mode behaviour. While the study highlighted a number of important robustness related parameters a methodology for identifying and understanding these as part of the system design would be highly desirable.

The design changes were made in response to the robustness issues in 88% of cases with these being predominantly software changes. Other common actions were specification changes (69%), verification method changes (62%) and FMEA updates (42%) which suggests these processes are not fully effective in preventing robustness issues. Design for robustness methods to be developed should therefore seek to augment or improve these.

The analysis of the earliest detection points suggests some opportunity to “left shift” detection to earlier in the development cycle before vehicles tests, particularly if viable model based testing approaches can be developed.

3.5.2 Discussion on Implications for Robustness Framework

Robustness concerns the resilience of a system to maintain a specified level of service despite errors caused by external faults and therefore is more about the ability to cope with these rather than preventing them occurring as this may not be possible, particularly in a Systems of Systems context. In a number of cases additional robustness was added into the system by design changes, e.g. defensive programming, after the occurrence of failures. There is a need to capture and classify such measures at a generic level such that they could be used as guidance in techniques at a software, hardware and system level to improve robustness.

While this would help at a bottom up-level there is a need for methods for a designer to be able to argue at a top down level whether their system is robust taking into account potential impacts from other systems.

The results give some cause for doubt as to the effectiveness of “desktop” analysis methods, e.g. specification reviews and FMEAs, in predicting robustness related failures. The potential robustness related issues are too numerous for exhaustive manual evaluation especially as they are often due to a complex combination of events (faults and activation patterns). This observation should be expected given that the failures in systems of systems are often emergent, and limitations in the effectiveness of FMEAs to detect complex issues are an expected shortcoming rather than a lack of rigor.

If predictive techniques such as FMEAs have limitations in detection of robustness failures in complex systems then it suggests that more empirically based techniques are required at an earlier stage in the development – i.e. there is a need to model and test. This in itself presents a number of key questions such as the selection of particular robustness related attributes to model and the ability to conduct modular testing on sub-parts of the system rather than model the system of systems as a whole. These reemphasize the need for the analytical techniques to determine important sources of variation in the system and to construct modular arguments for the robustness of a system.

Fundamental to the robustness of electronic control units is the continued availability of the processing platform and the data to implement functions. An area for investigation is robustness modelling of non-functional properties, e.g. voltage, initialization, and power-modes, relating to the *ability of the system to function* as opposed to the function itself.

Assuming such models can be constructed then the next consideration is the Design Of Experiment (DOE) method to be used to test them. In the case studies the test regimes were often updated as a result of failures particularly in relation to coverage and conditions. With a

number of independently variable parameters which are continuous in nature the test space explodes and, even with simulations, the experiment has to be constrained to match the time limitations. For such a design of experiments scalability is highly desirable such that the level of confidence increases throughout the testing. It is important to recognize that in the case of robustness testing, as opposed to functional testing, the test is not just looking for an expected response but for a much wider set of potentially undesirable results.

This case study has highlighted several potential directions for further work through understanding the nature of robustness issues. The next chapter will develop an overall framework for design for robustness and identify on specific method development required within this framework.

Chapter 4 - Development of Design for Robustness Framework

4.1 Introduction

This chapter utilises the understanding from the case studies into the actual causes of robustness issues in complex automotive electronics, both design and process related, and the knowledge of applicable techniques derived from the literature review to propose a Design for Robustness Framework for automotive electronics.

4.2 Design for Robustness Framework Requirements

In this section requirements for the Design for Robustness (DFR) framework listed based on the findings from the literature review and case studies.

[REQ1] The DFR Framework must support the system designers of individual systems and systems of systems during the design and development phases of a project.

[REQ2] The DFR Framework must support the specification of what service failures are acceptable as a result of external failures and how often they can occur in order to have a definition of what is acceptably robust.

[REQ3] The DFR Framework must provide a method to give assurance that the system can be trusted to be robust.

[REQ4] The DFR Framework must support an iterative and increment development approach.

[REQ5] The DFR Framework must provide a means for upfront risk analysis to determine priority areas where design effort to address systematic risks needs to be.

[REQ6] The DFR Framework must recognise the potential to apply formal methods to robustness issues.

[REQ7] The DFR Framework must provide a means of modelling robustness relevant properties at a useful level of abstraction.

[REQ8] The DFR Framework must provide the ability to model interactions between systems including fault insertion.

[REQ9] The DFR Framework must address transient faults originating outside the system affecting software which are typically types of communication faults including: race conditions, early and late interactions, erroneous or missing input signals, spurious communications or corrupted messages.

[REQ10] The DFR Framework must incorporate robustness related parameters of low supply voltage transients, ECUs shut down and initialisation, powermodes, and user operations.

[REQ11] The DFR Framework must cater for activation patterns including at least 3 factors.

[REQ12] The DFR Framework must provide a methodology for identifying and prioritising robustness related parameters as part of the system design.

[REQ13] The DFR Framework must provide a means of incorporating best practice design guidelines.

[REQ14] The DFR Framework must incorporate a means of systematic knowledge capture.

[REQ15] The DFR Framework must incorporate best practice test methods, including scalability such that the level of confidence increases throughout the testing.

[REQ16] The DFR Framework must be capable of detecting undesired results from verification tests not just expected response.

4.3 Overall design for robustness framework

An overall framework for design for robustness is proposed as shown in Figure 4.1. This framework starts with an initial design analysis based on experience, variance analysis and the development of robustness case. Beyond this an empirical approach is taken learning lessons from agile software development to iteratively deploy and test executable design artifacts from as early a stage as possible. In designing these test artifacts an input should be the design requirements identified in the robustness case as well as from robust design guidelines which should be embodied in standards, training and peer review processes.

The two items highlighted in orange boxes of robustness cases and robustness models will be the subject of development of new techniques within the thesis, the items in the grey boxes largely utilise existing techniques identified within the literature review.

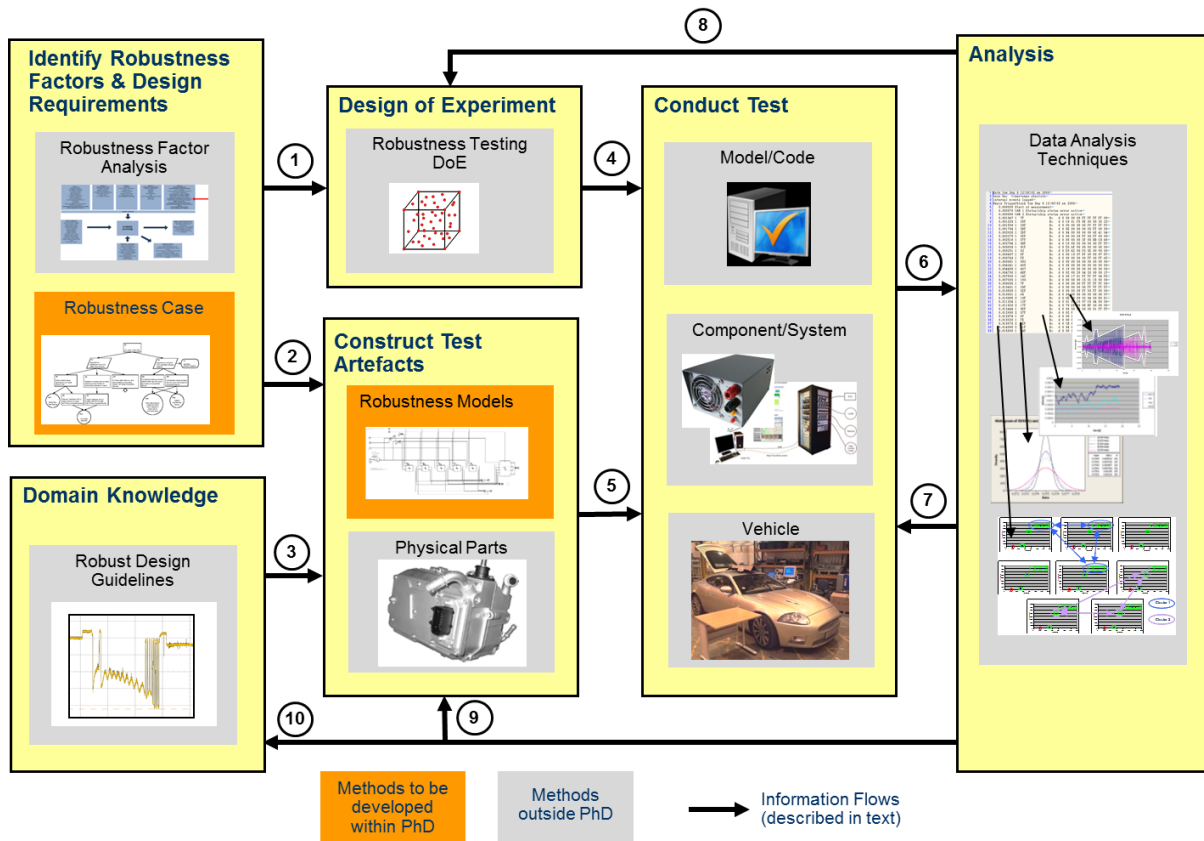


Figure 4.1. Framework for robust design of complex automotive electronics systems

The interactions between the individual elements shown by the numbered arrows are as follows:

1. Details of robustness factors and associated use cases for the design of experiment.
2. Design requirements to support the robustness case.
3. Design guidelines based on prior knowledge and experience of the domain.
4. Details of the design of the experiment to be carried out in the testing phase.
5. Models, code or physical parts to be tested.
6. Test results for analysis.
7. Requirements for re-testing based on analysis of test results.

8. Requirements for changing design of experiment based on analysis of test results.
9. Requirements for change to the design of the system under test based on analysis of test results.
10. New domain knowledge to be captured based on analysis of test results.

The intent is that this framework be used by system designers throughout the design & development of the system [REQ1] starting from that early concept phase of the project identifying robustness factors and design requirements. Then system developers progressively iterate through a design, implement, test, analyse and improve cycle, initially using models, then components and finally vehicles as development progresses [REQ4]. Throughout development domain knowledge should be captured and used to improve design guidelines. The individual elements are further described in the following section.

4.4 Design for Robustness Framework Elements

In this section the findings from the case studies are used to identify individual elements of the design for robustness framework. The techniques identified in the literature review are then used to develop methods within these elements with new techniques proposed to address any gaps. References are given throughout to indicate sources of further information on a particular method.

4.4.1 Design Guidelines

Design guidelines are required to systematically capture domain knowledge [REQ14]. In a number of cases additional robustness was added into the system after the occurrence of

failures by design changes, e.g. defensive programming. There is a need to capture and classify robust design measures at a generic level such that they could be used as guidance in techniques at a software, hardware and system level to improve robustness. As well as lessons learned from automotive experience, for which the DFR Framework creates a feedback mechanism from testing, design guidelines should incorporate best practice design guidelines the literature review highlighted some best practice from other fields such as the concepts of survivability (Knight et al, 2001), Self Stabilization (Arora and Gouda, 1994), and elementary interface design concepts (Kopetz, 1999) [REQ13].

There could be a number of complementary mechanisms for deploying such design guidelines. Specific requirements can be embodied as company design standards, which will ensure compliance in any system they are cascaded to. However a more effective mechanism may be via training whereby the wider *principles* of design for robustness can be taught as well as specific practices. A robust design principles and techniques course would fit well with modular technical training schemes. Another mechanism would be Design Guidelines documents that capture good and bad practice, though less formal than standards they can be more flexibly applied. Robustness cases are also a mechanism to capture knowledge which will be subsequently discussed. These mechanisms should be backed up by design reviews by peers and robustness specialists.

The guidelines should provide guidance on: the specification of acceptable service levels for degraded modes when faults are present [REQ2], fault tolerant design methods particularly against communication faults [REQ9] and measures to improve design robustness to parameters of low supply voltage transients, ECU shut down and initialisation, powermodes, and user operations [REQ10].

4.4.2 Identification of Critical Robustness Parameters

The case studies highlighted that there were certain parameters and operating modes which were more prone to cause robustness related failures. Examples of these were supply voltage variability, processor availability and system initialisation and shut down operating modes. While the case studies highlighted the difficulty in relying on desktop methods for identification of specific failure modes, a breakdown and analysis of robustness parameters to identify areas which are more susceptible to robustness failures would allow more design focus on these [REQ5] and specific levels of acceptable service to be specified against these [REQ2].

This variability analysis could be achieved through techniques such as review of previous design issues (the case studies serve as a basis for this) and variability focused design FMEAs. One practice of particular use is Parameter diagrams (P Diagrams) (Fritzsche, 2006). These break down noise sources in a system against those caused by; piece to piece variation, change over time, customer, external environment and system interaction [REQ10]. Those parameters deemed critical for robustness either due to experience failures or criticality of impact could then be the focus of particular design measures or scrutiny in validation [REQ12].

4.4.3 Robustness Cases

While design guidelines would help at a bottom-up level there is a need for methods that would enable the system designer to argue at a top-down level whether their system is robust taking into account potential impacts from other systems. A top-down method used in safety assessment is Safety Cases which are structured arguments concerning why a system meets its safety goals. It may be possible to use this approach in an analogous manner to argue why a system is robust.

Hence a Robustness Case would be a structured argument on why a system (of systems) is robust and is analogous to a safety case which “should communicate a clear, comprehensive and defensible argument that a particular system is acceptably safe to operate in a particular context” (Kelly and Weaver, 2004). The safety case should clearly argue how the evidence provided, e.g. design specifications, development methods and test results, demonstrates that the safety requirements of the system have been fulfilled. In a similar manner the Robustness Case gives a structured argument for how robustness requirements are met. This serves as a structured thought experiment for the system designers making them consider and explicitly express robustness objectives at an early stage of the design [REQ2/5]. It also gives confidence to decision makers that there has been adequate consideration of robustness in the system design and there is traceable evidence of how robustness objectives have been met [REQ3]. The literature review identified that safety cases often use a notation called Goal Structuring Notation (GSN) proposed by Kelly and Weaver (2004) and this could be adopted for Robustness Cases as it offers a visual representation of the argument structure which can be more readily understood. Benefits using safety arguments for modular systems (Bate, Hawkins and McDermid 2003), in that if modularity could be argued the safety case would hold for a variety of configurations, may also apply to certain automotive systems where there are standard elements e.g. network interfaces allowing degrees of modularity.

The argument structures can document best practice approaches to achieving particular robustness objectives [REQ14] to make design resilient to typical types of robustness failure [REQ9/10].

4.4.4 Design and Construction of Test Artefacts

The next step is to design and construct test artifacts system at various levels of abstraction, which initially will be models of the operating strategy at a high level [REQ7], before moving to more detailed models of the control and physical interactions and ultimately physical parts. In designing these test artifacts an input should be the design requirements identified in the robustness case and from robust design.

The case studies identified that an important factor for the robust operation of systems is the availability of the processing platform and the data to implement functions and the ability of the system to cope with spurious disruptions to these [REQ8]. Robustness modelling methods will be developed in the next chapters focusing on non-functional properties, e.g. voltage, initialization, and power-modes, relating to the *ability of the system to function* as opposed to the function itself [REQ9/10]. These models could be used for robustness analysis, formal model checking [REQ6] and as a platform model within a SIL or HIL test rather than assuming the control strategy can run at any time.

As well as more robustness orientated models encompassing the failure modes of processing and communication systems more robustness orientated models are required of physical systems to encompass the complete scope of variation within them rather than modelling them at a nominal state. Such advanced physical modelling methods and their application to HIL based robustness testing of control systems is described by Thanagasundram (Thanagasundram et al., 2008).

There is also a requirement for techniques to explore use cases, particularly for new systems. Modelling and simulation can be used to support earlier and wider user trials of new functions. An alternative approach may be to use formal analysis of models to determine potential situations whereby undesirable events can occur, i.e. reverse engineer use cases

[REQ6]. These would have to be plausibility checked to determine whether they could potentially occur.

4.4.5 Design Of Experiment (DOE)

The most suitable techniques identified in the literature review for the design of experiments for robustness testing were; structured pseudo random testing, Classification Tree Method and covering arrays [REQ15].

Monte Carlo or Pseudo random testing can quickly achieve wide coverage of the overall test space then fill it in at an increasing density. This approach has already been proved to be effective for dealing with robustness issues related to low voltages (Huang, McMurran, Dhadyalla and Jones, 2009). This work was based around a relatively limited number of parameters and used weighting of the pseudo random distributions to increase coverage in critical areas.

For more parameters the design of the experiment will have to be further optimized and should also utilize expert knowledge in order to do this. An approach may be to view the experimental design as an optimization problem to give a more dynamic or agile design of experiments whereby the experiment evolves based on previous results of random sampling. A structured approach to design of experiment for robustness testing using classification tree method and covering arrays is described by Dhadyalla (Dhadyalla et al., 2013) [REQ11].

4.4.6 Robustness Testing

Robustness testing should take place on the test artifacts at various levels of abstraction. Initially this will be on models using off-line test platforms, moving through hardware in loop testing at component and system level, before testing of vehicle level. Such a multi-level systematic approach to model-based testing is advocated in model based

systems engineering approaches (Forder, 2012). Test Automation is a key enabling technology to be able to conduct a large number of tests, allowing increased coverage and an ability to repeat testing at all change points [REQ11]. This can become more difficult as test artifacts become physical parts in terms of observability and may require development of bespoke test solutions such as video capture and image processing (Huang et al, 2009).

An alternative approach to verifying models in terms of robustness is to use formal methods [REQ6]. This approach would use model checkers based on formal methods to confirm that the important properties related to robustness hold. These properties could be based on the robustness goals identified within the robustness cases. There is collaborative work in projects such as the PICASSOS project (Ricardo, 2013) to mature formal methods in the context of verifying the functional safety of automotive systems which if successful will allow them to be applied in a robustness context.

4.4.7 Data Analysis

For such complex and large scale experiments into robustness it becomes important not only to be able to automate the testing but also to automate the analysis of data. Inevitably such experiments will create the requirement to analyze very large data sets and, as some robustness issues will not be discovered until the vehicle is in the field, this will also include field data.

Data analysis of the results of robustness testing can also be challenging not just due to the volume of data but also because the test is not looking for the expected response to a stimulus as in functional testing but rather whether an unexpected response has occurred, which by definition is more difficult to find [REQ16]. Therefore the use of data-mining tools is essential (Taylor et al, 2012). Ideally continuous measures of the health of a system and its components should be used as these can expose areas of the test space where levels of

variation or interactions cause risk without manifesting in customer level failures which then can be the subject of more focused investigation.

4.5 Conclusions and Next Steps

This framework identified represents a structured approach to design for robustness of automotive electronics. Many parts of the framework are based on state of the art practices identified within the literature review. Figure 4.2 summarises how the requirements for the framework set out at the beginning of the chapter are addressed by the various elements.

As well as integrating existing methods there are two new methods of robustness cases and robustness models identified in the framework which require further development. In the next chapter these methods will be developed through application to a fiber-optic based infotainment system to create guidelines for their generic application in terms of inputs, contents and structuring of the robustness case and models. Subsequently they are evaluated through trialing them by applying them to a hybrid electric propulsion system.

		Framework Element							
		Overall design for robustness framework	Design Guidelines	Identification of Critical Robustness Parameters	Robustness Cases	Design and Construction of Test Artefacts	Design Of Experiment (DOE)	Robustness Testing	Data Analysis
DFR Framework Requirements - The DFR Framework must.....									
[REQ1]	support the system designers of individual systems and system of systems (architectural level) during the design and development phases of a project.	X							
[REQ2]	support the specification of what service failures are acceptable as a result of external failures and how often they can occur in order to have a measure of what is acceptably robust.		X	X	X				
[REQ3]	provide a method to give assurance that the system can be trusted to be robust.				X				
[REQ4]	support an iterative and increment development approach.	X							
[REQ5]	provide a means for upfront risk analysis to determine where design effort to address systematic risks needs to be.			X	X				
[REQ6]	recognise the potential to apply formal methods to robustness issues.				X			X	
[REQ7]	provide a means of modelling robustness relevant properties at a useful level of abstraction.					X			
[REQ8]	provide the ability model interactions between systems including fault insertion.					X			
[REQ9]	address transient faults originating outside the system affecting software which are typically types of communication faults includes race conditions, early and late interactions , erroneous or missing input signals, spurious communications or corrupted messages.		X	X	X	X			
[REQ10]	incorporate robustness related parameters of low supply voltage transients, ECUs shut down and initialisation, powermodes, user operations.		X	X	X	X			
[REQ11]	cater for a variety of activation patterns including at least 3 factors.						X	X	
[REQ12]	provide a methodology for identifying and prioritising robustness related parameters as part of the system design.			X					
[REQ13]	provide a means of incorporating best practice design guidelines.		X						
[REQ14]	incorporate a means of systematic knowledge capture.		X		X				
[REQ15]	incorporate best practice test methods, including scalability such that the level of confidence increases throughout the testing.						X		
[REQ16]	be capable of detecting undesired results from verification tests not just expected response.								X

Figure 4.2. Design for Robustness Framework elements vs. requirements

Chapter 5 – Development of New Robustness Techniques through Application to an Infotainment System

5.1 Introduction

In this chapter the two novel techniques, specifically robustness cases and robustness modelling, are developed through application to a fiber-optic network based infotainment system. Firstly an overview of the MOST (Media Oriented System Transport) fiber-optic infotainment network is given as background to give an understanding of the technology and associated issues. Following this a robustness case for a MOST infotainment system is developed based on safety case practices identified in the literature review. The next section describes the development of robustness models, initially as a generic approach before describing application to the Control Channel of a MOST infotainment system. Initial conclusions are made on the usefulness of these two new techniques. Finally the generic content of the core methods is abstracted and documented.

5.2 MOST Infotainment Systems

MOST (Media Oriented System Transport) is a communication system with a flexible architecture designed specifically for automotive infotainment systems. This section gives an overview of MOST, a more in-depth description is given in “MOST - The Automotive Multimedia Network” (Grzemba 2008). The detailed specifications for the network and network interface controller used in this work are contained in MOST Specification Rev 2.4 (MOST Cooperation, 2005) and OS8104 MOST Network Transceiver Final Product Data Sheet (Oasis, 2003).

Growing infotainment system complexity, particularly multiple audio and video channels causing excessive amounts of wiring, lead to the need for a high bandwidth network. The key requirements for this network are that it should be capable of the distribution of multiple digital audio channels, have a high immunity to noise and be highly flexible and reconfigurable. As a result vehicle manufacturers and suppliers formed the MOST Cooperation to develop common standards for such a network. MOST was first applied to the BMW 7 Series in 2001 and is now used by BMW, Mercedes, VW/Audi, Toyota, Volvo, Porsche, Jaguar Land Rover and Hyundai in over 100 vehicle models. MOST is now available in three bandwidth variants, 25Mbits/sec (MOST25), 50Mbits/sec (MOST50) and 150Mbits/sec (MOST150), with the increase in bandwidth supporting video networking and high definition audio.

MOST is effectively a composite bus able to transmit three types of information with different requirements. The first is control data necessary for command signals, status signals and small packet data. This needs comparatively limited bandwidth, message arbitration and low latency. The second data type is synchronous data for continuous real-time transmission of streaming audio and video files. This needs large amounts of guaranteed bandwidth with minimal buffering. The third data type is asynchronous data for transmission of sporadic bursts of ‘large’ packet data e.g. contacts lists, navigation data or web pages. This needs intermittent access to a variable bandwidth channel.

To achieve a sufficiently high bandwidth, noise immune communication MOST typically uses an optical medium to transfer data with an eye-safe LED light-source linked by Plastic Optical Fiber (POF) in a ring topology. To accommodate different types of data the MOST data frame is split into three sections with an ability to dynamically re-apportion bandwidth between synchronous and asynchronous data. The MOST topology and frame structure is illustrated in Figure 5.1 below.

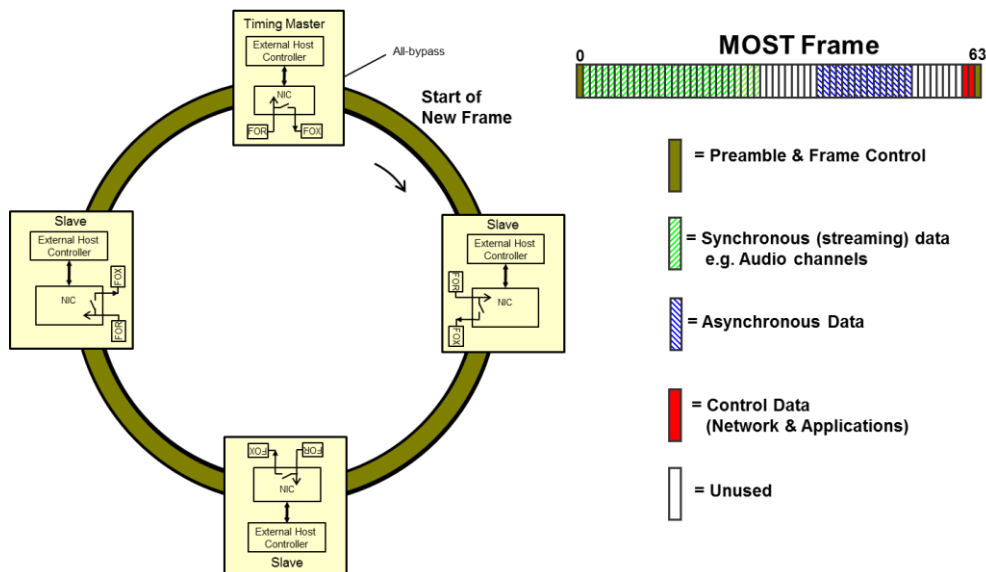


Figure 5.1 MOST topology and frame structure

The operation of the bus is controlled by the bus master which generates a frame that travels from node to node around the ring. Each node receives information and updates any of its own information with a 2 bit delay per node in transmission. Each frame is 512 bits so the bus master receives back the start of the frame while still sending the frame.

Each MOST node has a “Network Interface Controller” (NIC) which provides the low level interface to the bus via a Fiber Optic Transceiver (FOT). Each node must also implement a standard set of communication software called MOST NetServices which control communication from the application software to the network interface controller. This abstracts control to application level peer to peer communications via standard interface definitions known as Fblocks (short for function blocks). It is vital that the network interface controllers and NetServices implemented on all nodes are compatible.

For the robustness of MOST-based infotainment systems the initialisation is a key phase which is influenced by the status of the individual MOST nodes and the interactions between them on the control channel. MOST allows features to be dynamically registered, typically at start-up, through a process of each node registering its function blocks and then

compiling and distributing a registry of which functions each node contains. Following this the nodes can set up peer-to-peer interfaces. If new functions are registered or de-registered during or after the initialisation nodes are notified of this configuration change but this can be disruptive during initialisation.

Hence initialisation is the busiest period of MOST control channel communication, and becoming increasingly challenging to optimise as the number of customer features, and therefore number of MOST function blocks, increases. There are key customer requirements to satisfy, not only that the system starts up in a robust manner with continuity of correct service (i.e. reliability), but that certain infotainment and comfort features are available as soon as the vehicle is started (i.e. availability).

Types of service failures which could occur and must be mitigated in the system design include; excessive delay before audio sources are available, missing customer features, and delayed system startup. Traditionally, these types of failures are driven out during system practical testing and signoff. This can be a long resource consuming process, even with recent advancements in automated testing, and will only become more complex as feature numbers increase.

Infotainment systems must be designed to be highly modular and reconfigurable to allow varying levels of features to be offered and cater for variations between markets e.g. navigation systems and tuners. This presents an additional challenge in ensuring the system is robust in all configurations.

5.3 Infotainment Robustness Case Development

In this section a robustness case is developed for an infotainment system analogous to a safety case based on methods for safety case construction identified in the literature review.

The development of the infotainment system used for the study was already underway and so the objective was to ensure the set of specifications and requirements were sufficient to meet robustness objectives. At the end of the chapter the generic principles behind the approach and development steps are captured.

5.3.1 Goal Structuring Notation

The diagrammatic technique Goal Structuring Notation (GSN) was selected to construct the robustness case for infotainment as the approach offers benefits in terms of clarity and structure (Kelly and Weaver 2004). Of particular importance in the context of complex systems is that GSN can be used to hierarchically decompose robustness objectives into a number of sub-arguments which have the potential to be reused as design “patterns” (Kelly and McDermid 1997).

Goal Structuring Notation has symbols representing Goals, Solutions, Strategies, Contexts, Assumptions and Justifications with arrows representing relationships between them. Arguments are constructed in a hierarchical form and strategies can be used to describe the approach for decomposing a high level goal into sub-goals. The GSN Community Standard (GSN Community 2011) gives a comprehensive and authoritative definition of the Goal Structuring Notation and best practice guidelines on its use. A short overview of the key constructs is given below through description of an example.

Figure 5.2 shows an example of part of a GSN safety argument. At the top is a safety goal that hazard occurrences occur less than defined targets, in this case it is actually a sub-goal within the overall safety case. This goal is shown by the sideways linking arrow with the hollow arrowhead to be in the particular context of risk targets for hazard occurrence. The arrow downwards with the solid arrowhead shows how the goal is supported by a strategy of using industry standard controllability targets which is in the context of these industry

standards. The strategy is supported by a single goal in the example though it is typical that a strategy decomposes a goal into a number of sub-goals. The goal in this case, is that the Markov chain failure probability modelling of the items on the fault tree analysis meets these targets. In this case an assumption is highlighted that the fault tree analysis is comprehensive which notes an unsubstantiated statement. This assumption must be followed up to ensure the safety case is robust. It is also possible to add justifications to the diagram to give evidence for assertions. Finally the goal is shown to have a solution, marked as a circle, of the fault tree and Markov probability analysis that would be the evidence provided to show the goal has been met. In this case it is a single solution but it can be multiple solutions which must all be done to meet a goal or alternative solutions which are individually sufficient.

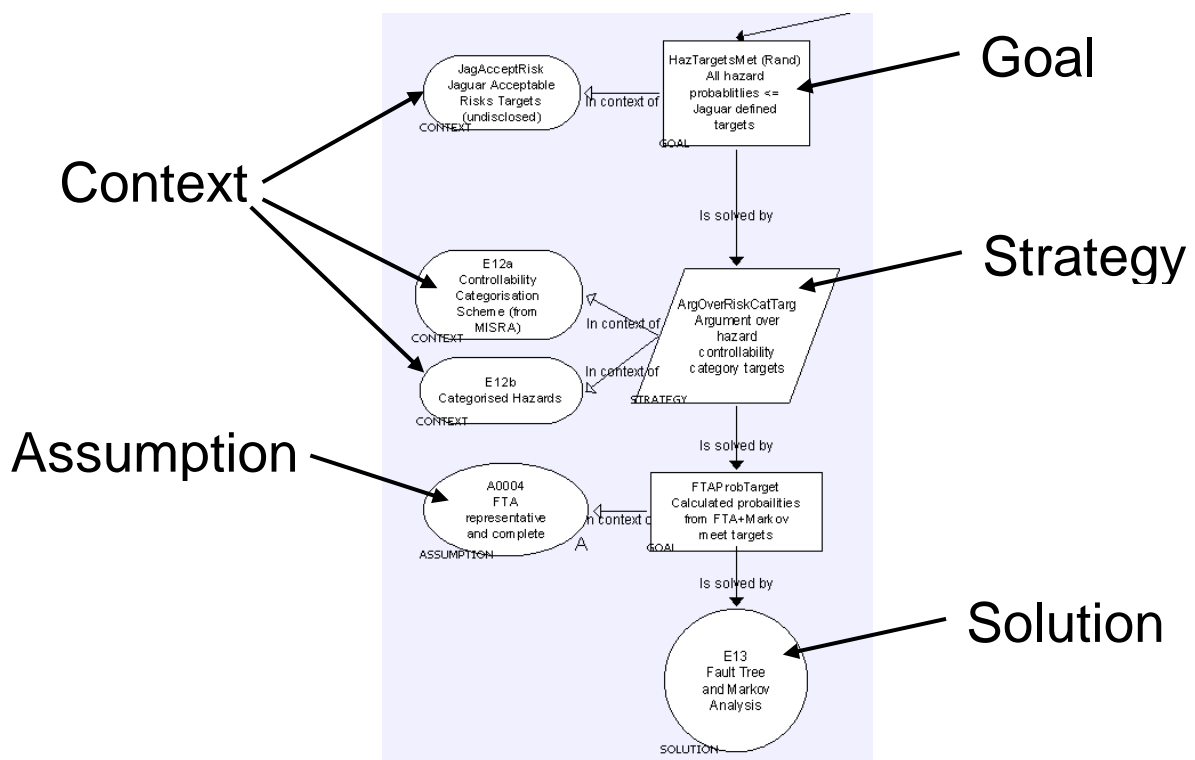


Figure 5.2 Example of Goal Structuring Notation (based on Barker, Kendall and Darlinson 1997)

The safety case for complex systems can be decomposed into a hierarchical argument consisting of a number of modules which can be re-used in similar contexts (Kelly 2001, Bates, Bate, Hawkins, Kelly, and McDermid, 2003). This allows for separation of concerns while maintaining a coherent overall argument and enables the identification of goals in discrete argument modules which are applicable at system of system, system and component levels. To compose different arguments modules together, the goals required and fulfilled by other modules must be consistent. For example the contexts assumed by different modules must be shared. When a successful composition has been achieved an output is a contract which records the agreed relationships between modules. Changes to a module argument thereafter then must adhere to the conditions of the contract.

Kelly goes on to list the key principles behind the architecture of a modular safety case as:

- High Cohesion and Low Coupling
- Supporting Work Division and Contractual Boundaries
- Supporting future expansion
- Isolating change

Some of the inferences from this are a separation of software and hardware arguments, separate arguments for highly dynamic parts of systems, avoiding unnecessary restrictions of context and care in-cross referencing and definition of goals which should be as solution independent as possible. Figure 5.3 shows an illustration of the partitioning of a Modular Safety Argument Structure for Integrated Modular Avionics given in Kelly, 2001.

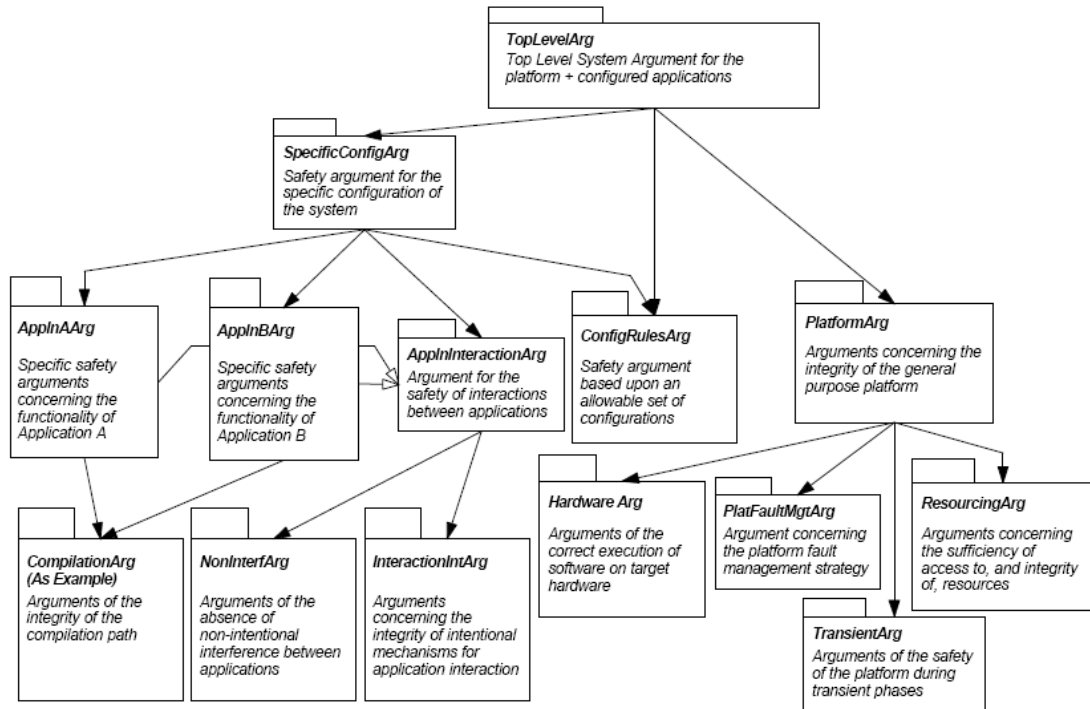


Figure 5.3 Modular safety argument structure for integrated modular avionics illustration of safety case partitioning (Kelly, 2001)

Palin and Habli (2010) set out to define an approach to create a “safety case catalogue” using GSN patterns and modular extensions capable of constructing a full vehicle safety case. The high level argument for vehicle safety is split into strategies for assuring acceptable safety during and after product development. The strategy for assuring safety during product development is supported by two types of sub modules (“away goals”): one for meeting pre-defined system safety requirements e.g. crash performance regulations, and a second type for system safety which has instances for each vehicle system. The strategy for assuring post product development safety is supported by two sub-modules one relating to production and the other relating to servicing, in-use monitoring and maintenance.

For the product development away goals top level argument patterns were developed. The System Safety argument was expressed as a "Risk Management Argument Pattern"

which had three Away Goals, i.e. sub modules, justifying the processes used for Hazard identification, the specification of safety goals and for the creation of the FMEA. The case study was conducted on a stop/start system and the instantiation of the 'Risk Management Argument Pattern' and 'Risk Mitigation Argument Pattern' were illustrated.

The authors observed this approach led to better argued and more traceable safety cases and noted the modular structure was a good method of linking design artifacts e.g. state, sequence and logic diagrams to the safety case. The authors also stated there were a number of limitations but did not expand on these other than to state it was necessary for those constructing the safety case to be familiar with the domain.

5.3.2 Application of Goal Structuring Notation to Robustness Cases

The first concern in the development of the Infotainment GSN robustness case was the selection of a suitable tool. There are a number of tools available; the GSN Working Group (2012) lists the following:

ASCE (Adelard Safety Case Editor) Adelard www.adelard.co.uk

E-Safety CasePraxis HI S www.esafetycase.com

GSN CaseMaker ERA Technology www.era.co.uk

ISCADE (Integrated Safety Case Development Environment)RCM2 www.iscade.co.uk

ISISHigh Integrity Solutions www.highintegritysolutions.com

University of York Freeware Visio Add-on - University of York

After comparing with a bespoke GSN modelling tool (ASCE) it was decided to use Microsoft Visio with the University of York Freeware Visio Add-on as this offered the required functionality in a readily available and easy to use tool. In particular the University of York plug-in offers good support for modular arguments. For an industry roll-out of this

technique it would be necessary to revisit tool selection based on: the tools already used in a particular organisation, the ability to link to other tools such as requirements management tools and the ability to export GSN as a hyperlinked HTML file offered by some commercial tools.

The first step in developing a GSN safety case is to define: the high level safety goal, the system and the related context to which it applies, so a similar approach was taken for the robustness case. For the infotainment system robustness case the top level goal was defined as *“Infotainment system is acceptably robust in normal operation”*. To provide clarification on the particular system a contextual statement was linked to defined the system context as *“Individual Systems within GEN2.1 Phase 3 MOST Infotainment System”*. While the definition of the system is suitably precise it was recognised further steps would be required to define the context of normal operation and to justify what was acceptable robustness. However for this case study the emphasis was on developing the argument structure so it was decided to focus on this rather than further resolving these application specific definitions. Hence there was a need to state two assumptions regarding this goal that would need to be followed up and clarified. The first was that normal operating conditions are defined in the system specification. The second was that acceptable robustness is defined in the system specifications, i.e. the acceptable levels of service during external failures. This top level argument is shown in Figure 5.3.

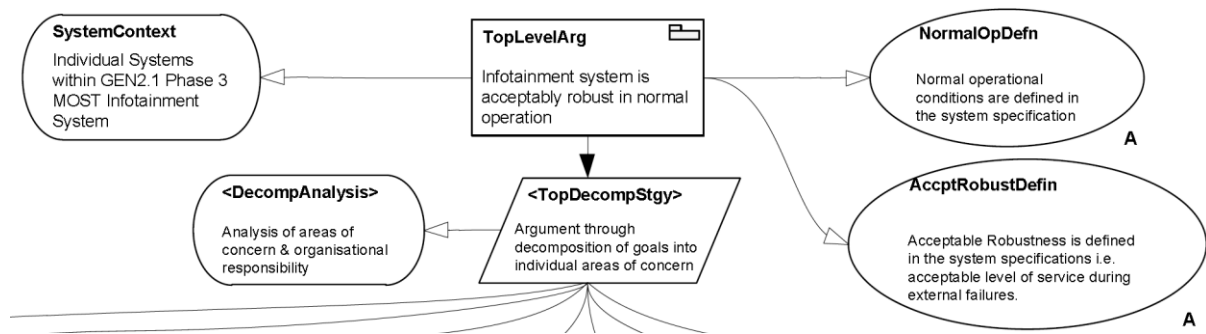


Figure 5.3. Infotainment system robustness case top level goal

Having identified the top level goal the next step was to develop a strategy to decompose this into a number of sub-goals which would form the basis of argument modules. Using the previously noted principles of high cohesion and low coupling, supporting work division and contractual boundaries, supporting future expansion and isolating change a proposal for the decomposition of the infotainment robustness case was developed and is shown in Table 5.1.

Argument	Responsibility		Description	Generic/ Specific (Instantiation Required)
	OEM	Supplier		
Software				
Operating system		✓	Argument for robustness of operating system	Derivatives for operating systems
Functions		✓	Argument for robustness of implementation of specific functions	Specific for function
Function Interaction	✓		Argument for robustness of interaction of function with others	Specific or overall argument?
Hardware				
Application Hardware		✓	Argument for robustness of node hardware	Specific for node
Communications channels				
MOST Physical layer	✓		Argument for robustness of MOST Physical Layer	Generic
NIC / INIC	✓		Argument for robustness of Network Interface Component	Specific
Netservices	✓		Argument for robustness of Netservices	Generic
Application layer		✓	Argument for robustness of application layer interface to netservices	Specific for node
Transient Behaviour				
Initialisation	✓	✓	Argument for robustness of initialisation behaviour	Generic
Shut down	✓	✓	Argument for robustness of shutdown behaviour	Generic
Error Recovery				
MOST	✓		Argument for robustness of MOST error recovery	Generic
Component		✓	Argument for robustness of component error recovery e.g. watchdogs	Specific
Unintentional Interaction				
Unintentional Interaction	✓	✓	Argument for robustness against unintended interactions	Generic?
Configuration Argument				
Configuration Argument	✓		Argument for robustness of configurability of system	Generic

Table 5.1 - Decomposition of the infotainment robustness case into argument modules

The intent of this approach was to recognise work division and contractual boundaries particularly between the OEM and the supplier and where different parts of the organisation specify aspects of the system behaviour at hardware, software and communications levels. In addition, noting areas of high risk identified by the case studies, specific argument modules were added for transient behaviours (initialisation and shut down), for error recovery and for unintentional interactions. Finally, recognising this was a modular system, an argument was added concerning the configurability of the system, i.e. justification that the system would be robust in any potential configuration. This was used to define the top level argument structure breaking down the top level goals into sub-goals with associated argument modules shown in figure 5.4.

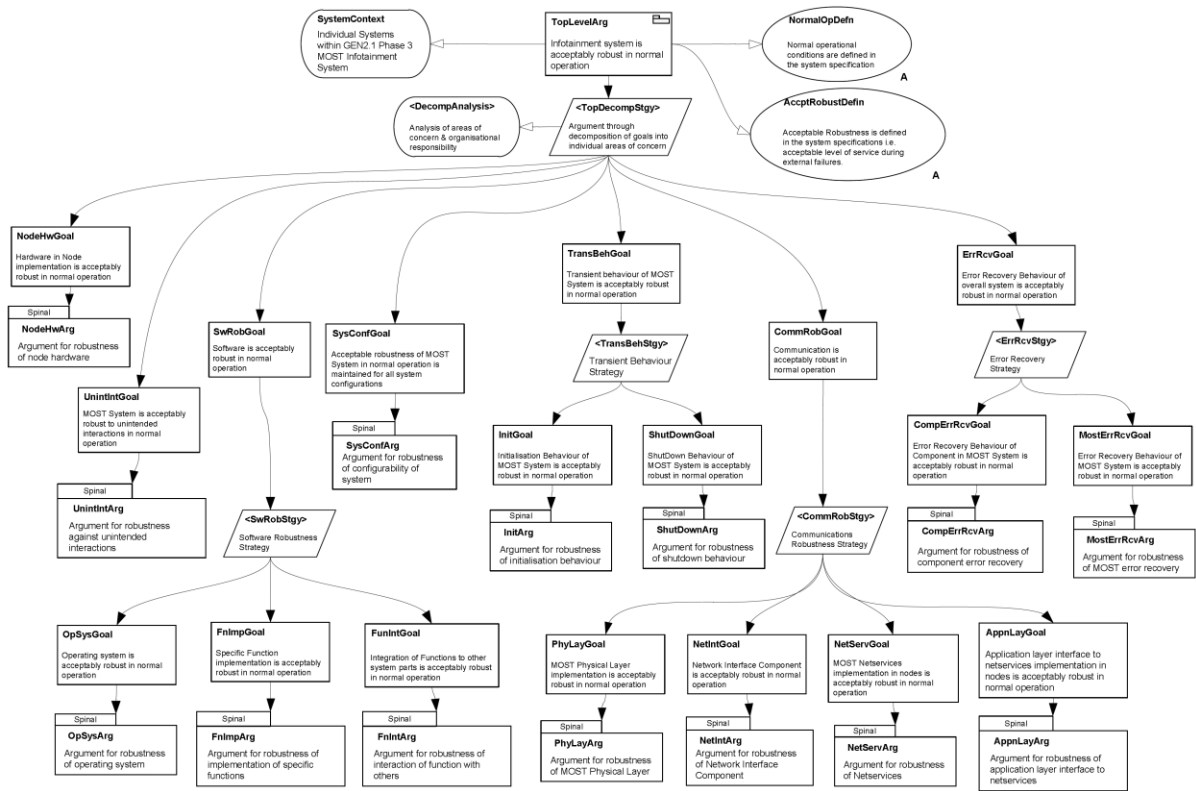


Figure 5.4 Infotainment system robustness case top level argument structure

Each of the argument modules identified in the top level argument was developed by breaking down each argument specific top-level goal into strategies, sub-goals and solutions which would form the evidence for achieving each sub-goal. These required domain knowledge to create and in this case were developed largely bottom up using information provided by relevant technical experts. However this did show that the person constructing the robustness argument did not have to be a domain expert providing they had adequate knowledge of automotive electronics and access to domain experts.

An example of one of these argument modules for the network interface controller is shown in Figure 5.5 with the full argument being contained in Appendix 1. The argument module shown illustrates the decomposition strategy for the robustness of the network interface into; selection of approved components, using correct version of firmware, 3rd party review, device level and system level verification testing. The solutions for these are the

evidence of supplier certificates of compliance, results of 3rd party review and results of supplier component tests and in-house system tests. In some cases a part of the strategy to meet a goal would refer to an existing solution in another argument module.

In constructing these arguments it helped identify where there were adequate measures in place and where there were gaps. This was possible through review of the argument to determine completeness and whether the solutions proposed were adequate in meeting the goals. One example was in the case of the network interface controller argument, where on the basis of the proposed solutions it was questioned whether the system Design Verification Method (DVM) had adequate robustness testing. In future application the argument should be the initial basis for robustness design reviews at an early stage in development to highlight and address such risks.

The robustness argument also highlights the need to follow up gathering evidence of compliance, both internally and externally, with each sub-goal rather than trusting that this had been done.

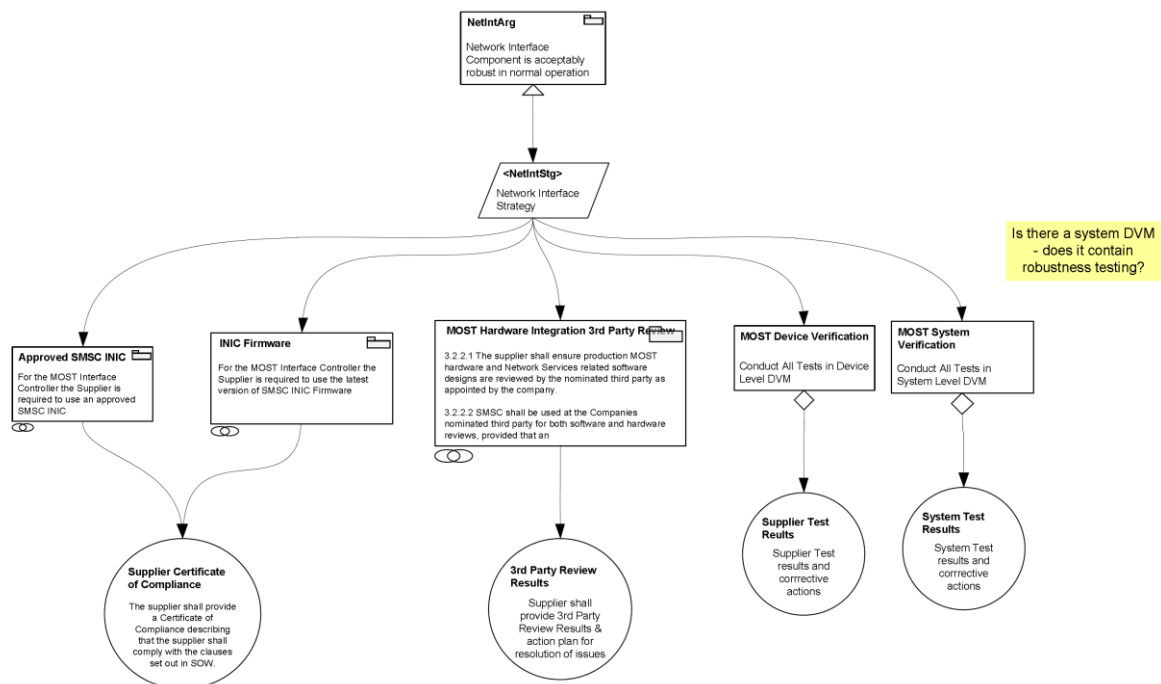


Figure 5.5 - GSN robustness argument module for the Network Interface Controller

5.4 Robustness Modelling

This section describes the approach taken to develop robustness models for distributed automotive electronics systems. These models were initially developed at a generic level in MATLAB Simulink and Stateflow to be able to capture both continuous and state based behaviour. Subsequently these models have been targeted at a specific application of the initialisation of a MOST infotainment system. The objective of this modelling was to be able to simulate the initialisation process to understand the critical parameters to the robustness and consistency of the process and to be able to investigate alternative strategies. At the end of the chapter the generic principles behind the approach and guidelines for model development are captured.

5.4.1 Approach To Modelling

The first concern with the development of any model is the level of fidelity to the real system to make it realisable and useable while allowing behaviour in the areas of concern to be sufficiently representative to make valid conclusions. In particular it is necessary to decide which attributes should be embodied in the model, the level of detail to which they must be modelled and which attributes can be left out in the abstraction of the model.

From the study of robustness related failures of automotive electronics it was seen that important factors are those related to the availability of the processing platform and the timely availability of data to implement functions from other systems. Hence it is necessary to model non-functional (to the customer) properties, e.g. voltage, initialization, and power-modes, relating to the *ability* of the system to function.

This focus on the ability of the system to function as opposed to the function itself then allows the effort to construct the model to be considerably reduced if the functional behaviour can be abstracted. The initial approach for doing this was to consider functions as operational if the processing platforms to implement it were available and the other functions providing input signals were available.

At a Systems of Systems level for automotive electronics it is necessary to capture continuous behaviour, such as the response to a varying cranking voltage, as well as state behaviour as robustness failures can be a combination of both. It was therefore determined to model the continuous behaviour in MATLAB Simulink and the state behaviour of an individual ECU (Electronic Control Unit) in Stateflow. These models were initially developed at a generic level based on five ECUs with a variety of initialisation characteristics connected via a CAN bus to explore the modelling approach.

5.4.2 Development Of Generic Model

A generic model was iteratively developed; the resulting top level structure is illustrated in Figure 5.6. The model consists of a number of individual blocks representing the ECUs, an operation mode block controlled by an engine start-stop switch which in turn controls supply voltage generation blocks and the powermode of the ECUs. Powermodes are used on vehicles where the key controlled states are replaced by electronically generated ones for power-off, accessory, ignition on, and cranking. The ECU blocks state machine has a timing input which can be phased between ECUs to represent the differences in timing that will naturally exist between ECUs. The supply voltage generation blocks can generate permanent supplies, ignition fed supplies and configurable cranking voltages. A final block determines whether distributed functions are operational.

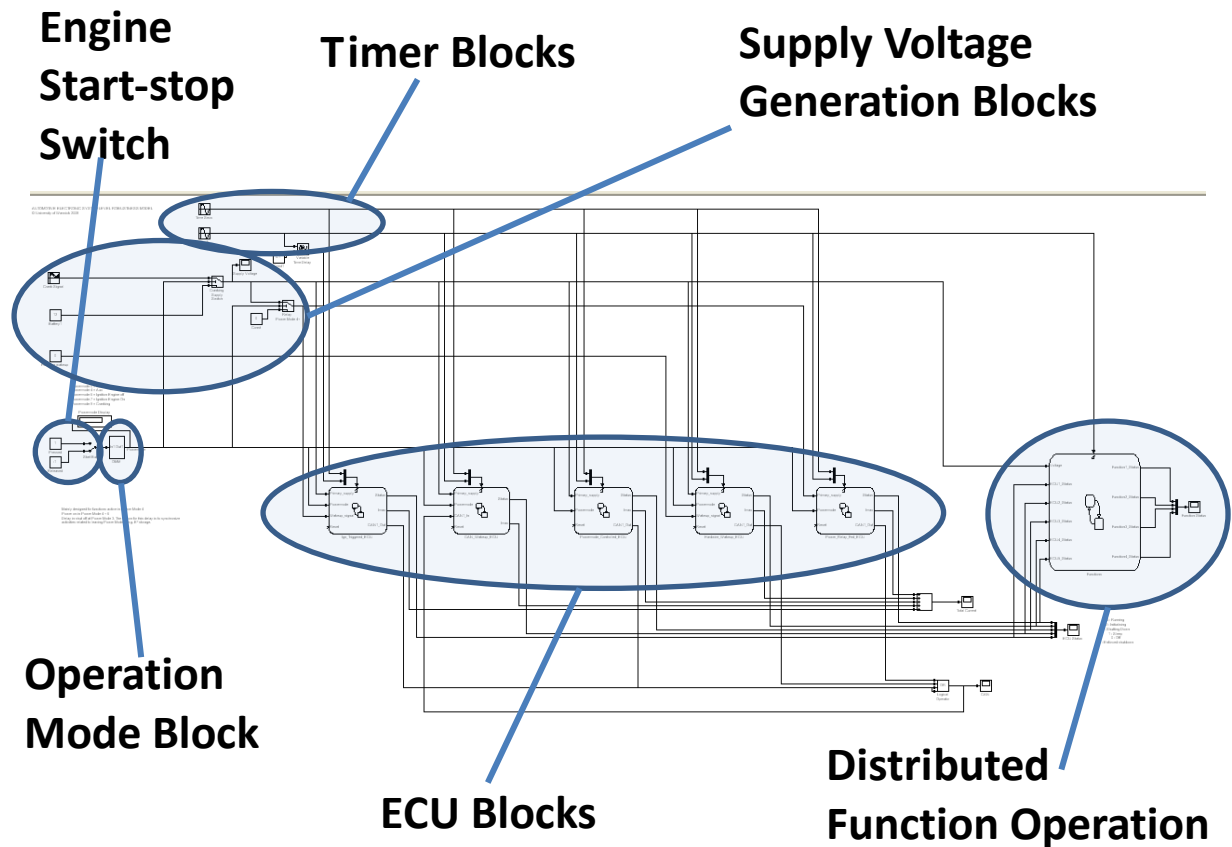


Figure 5.6 Top level structure of generic model

A variety of ECU blocks were constructed to represent different scenarios by which ECUs may be activated including supply voltage activated, powermode wake-up, CAN wake-up and hardwired signal wake-up. A more detailed view of the inputs and outputs to one of these blocks is shown in Figure 5.7. The inputs are supply voltage, a powermode signal which determines its operational state, and an external wake-up signal which could be a CAN signal, a hardwired signal or none in case of an ECU which is always active when powered. The minimum operating voltage at which it will go into an unpowered state if the supply voltage drops is configurable for each ECU. There is also an input which allows the block to be externally re-set to mimic a sporadic failure in individual ECUs to allow the effects on the overall system to be observed.

The outputs from the ECU blocks are Status, I_{max} and CAN_Out. Status is an integer value which relates to the current operational state of the ECU as determined by its internal state-machine, I_{max} is the potential current drain of the ECU in its current mode and CAN_Out is a binary value indicating whether the ECU is maintaining network communications.

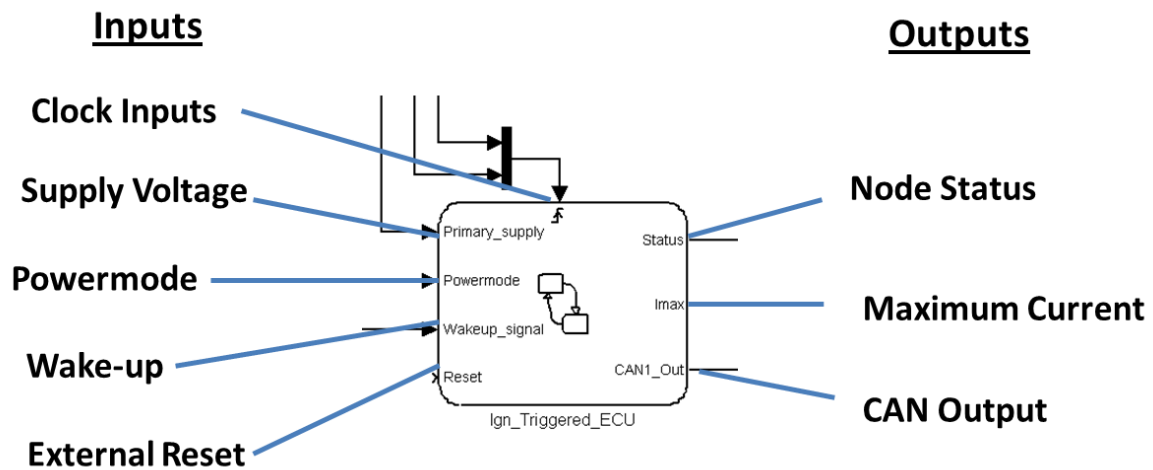


Figure 5.7: ECU Block from Generic Model

Within each ECU block is a state machine with the key operational states, the structure of this is illustrated in Figure 5.8. The main states are Unpowered, Sleep, Initialisation, On, and Shutdown. The state transitions are determined by the external wake-up inputs, the power supply level in relation to the minimum operating voltage, the external reset signal, and the Initialisation and Shut-down times which are configurable for individual ECUs.

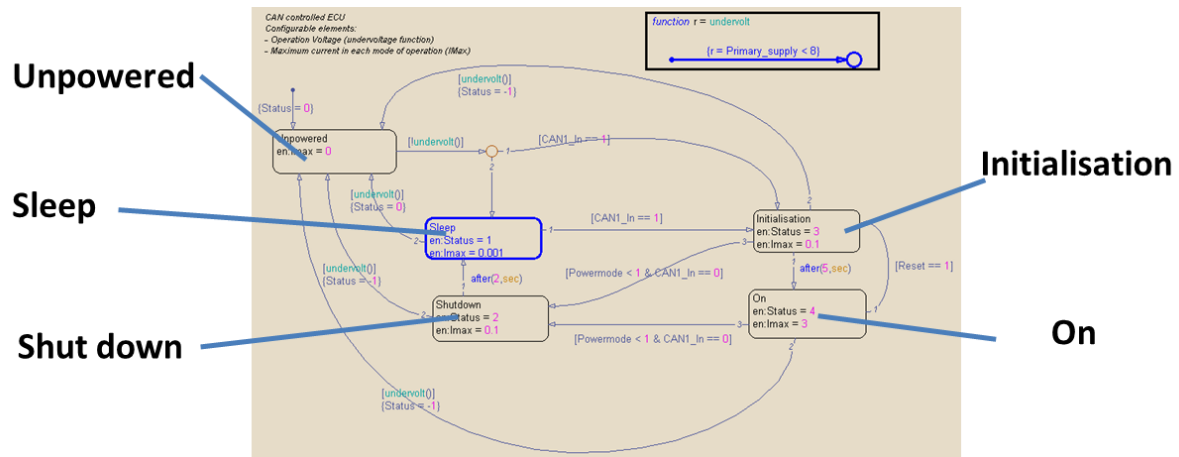


Figure 5.8: State Machine within ECU Block

An objective of the model is to show when distributed functions are operational. A block to determine this was added, a section of which is shown in Figure 5.9. Whether a particular function is active is based not only on the availability of the relevant ECUs, determined by status outputs from the ECU blocks, but also the availability of signals provided by other functions. When a function becomes active it sets flags to indicate that the signals it outputs are available to other functions.

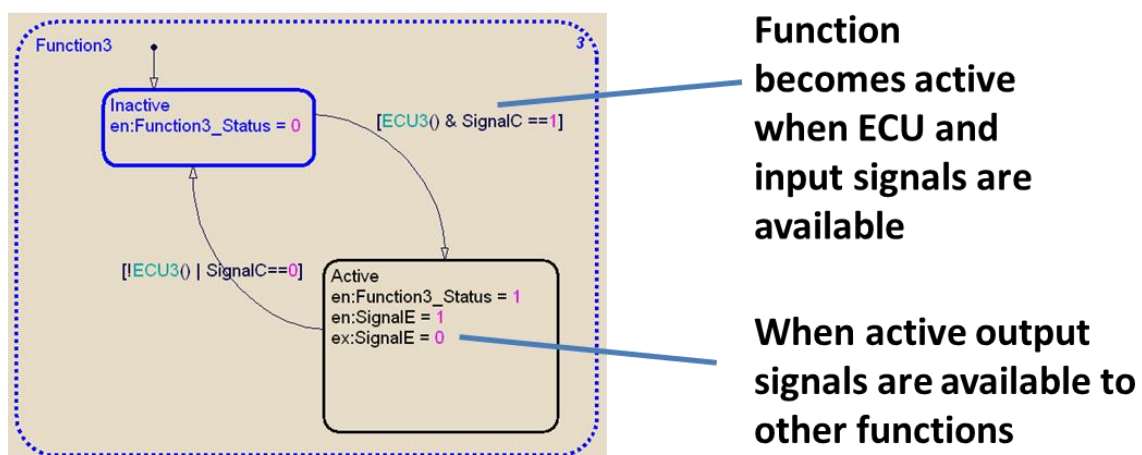


Figure 5.9: Part of the state machine for distributed function availability from generic model

5.4.3 Results From Generic Model

A key source of issues for the robustness of distributed automotive systems is from low voltage transients such as when cranking the engine with a battery in a low state of charge. In such situations at critical voltages the operational states of different parts of a distributed system may change highly dynamically and in a non-uniform fashion across the system giving the potential for operational issues. Novel approaches to developing practical tests on physical systems have been developed (Huang et al, 2010) but it would be desirable to use simulation to test such conditions, leading to early detection and resolution of issues. Hence a simulation of a low voltage crank was used to test the modelled system, examples of the results are shown in Figure 5.10.

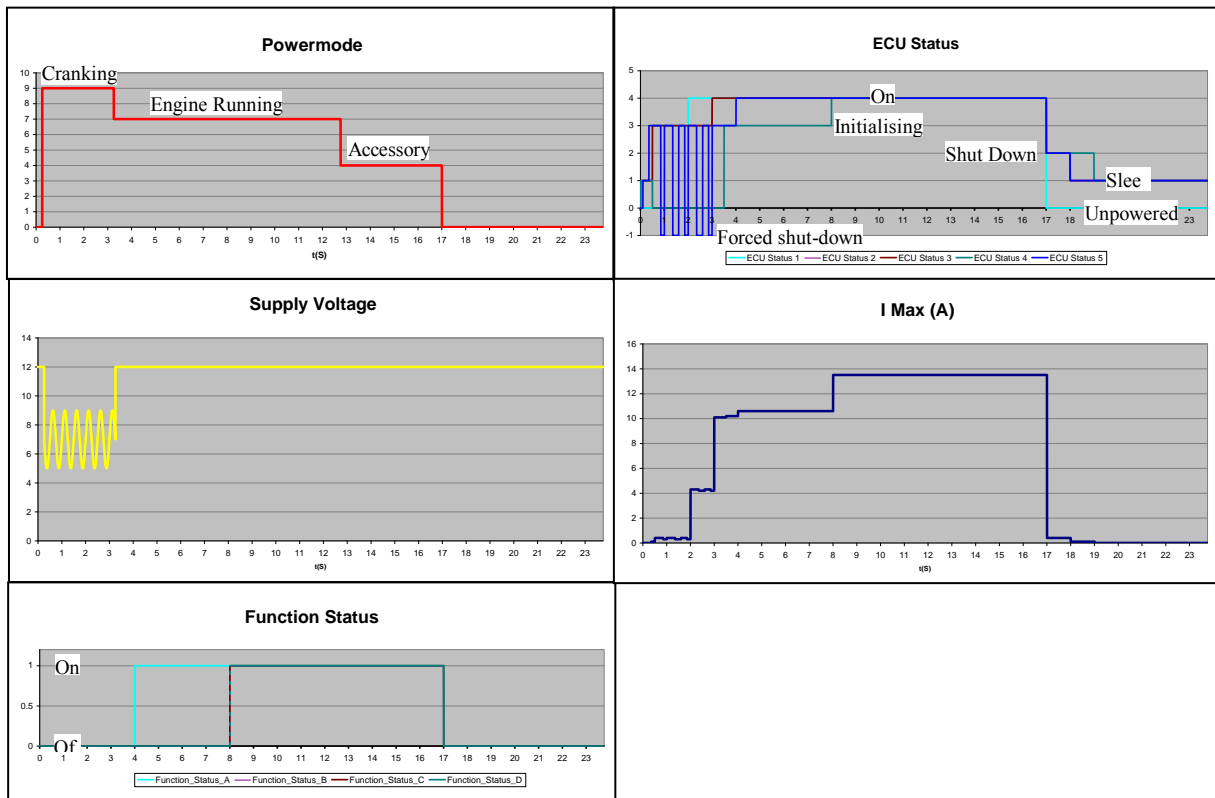


Figure 5.10: Simulation of low voltage cranking on generic model

In the test scenario shown in Figure 5.10 the engine start switch was pressed and held triggering a 3 second engine crank and then pressed again after 10 seconds to shut off engine, entering accessory mode for 4 seconds before going into power-off mode shutting down all ECUs as shown on the powermode graph. The supply voltage graph in Figure 5.10 shows a typical sinusoidal voltage dip when the engine is cranking, which recovers after the engine fires. It is possible to vary this signal pseudo-randomly to test a variety of scenarios.

The ECU status graph shows the resultant operational state of the ECUs illustrated by an integer value (0 = unpowered, 1= Sleep, 2 = Shutdown, 3 = Initialising, 4 = Running). A value of -1 is used where an ECU has gone unpowered without going through full shutdown which can cause anomalous behaviour. In the test scenario the crank signal has provoked this state in one of the ECUs. The ECUs can be seen to have varying times for initialization and shutdown. One ECU does not initialize until around 8 seconds the effect of this in delaying operation of functions can be seen in the functional status graph.

The graph marked I Max in Figure 5.10 shows the combined maximum current draw of the ECUs. This figure is of interest in maintaining voltage stability and in the detection of quiescent current issues, caused for example by nodes not going to sleep and keeping other nodes on the network awake.

While the results for this scenario with a limited number of interacting nodes and functions are readily understandable, as the complexity increases to that of real systems, they become less intuitive. The next step has been to apply the modelling principles shown to a specific application.

5.4.4 MOST Control Channel Model – Initialisation Robustness Issues

The specific application targeted was the initialisation of an infotainment system based on a MOST (Media Oriented System Transport) fibre optic network. The objective of this

modelling was to be able to simulate the initialisation process to be able to understand the critical parameters to the robustness and consistency of the process and to be able to investigate enhanced alternative strategies.

Analysis of MOST initialisations has shown a number of interesting behaviours. During high initialisation bus loading, cases have been observed where multiple slave modules are sending control messages to the master module simultaneously, leading to a high number of Non-Acknowledged messages (NAKS) which must be re-sent. During this period of intense retries it was possible for one slave module to monopolise the master module where only its messages were being received and acknowledged. This prompted the following questions to be investigated through the modelling. Why is the network services cycle time, the rate at which the network services software task is called by the module's operating system, the same in all nodes and could varying them improve initialisation timing? Why is the low level retry number, the number of times the Network Interface Controller will attempt to re-send a message, set to the same value in all nodes and could varying them improve initialisation timing? Why is the low level retry timing identical in all nodes and could varying them improve initialisation timing? During the notification setting phase, all modules are trying to set notifications simultaneously. This causes high MOST control channel bus loading, therefore high numbers of not-acknowledged messages, leading to initialisation delays.

Measures of network performance during start-up include: the time from light on until the system is fully initialised with all notifications set and audio sources set up, the time from light on until a particular customer feature is available and the number of MOST message low level and high level retries used.

Adjustable parameters which affect initialisation performance include: the Network Services cycle time, the number of low level retries and their timing, the use of application high level retries, and associated timing parameters, the receive and transmit message

buffering strategy, the MOST ring order, the scheduling of notification setting, and therefore order of feature availability. These parameters are not fixed in the MOST specifications and are therefore need to be specified by the vehicle manufacturer or module suppliers. Some of these parameters such as number and timing of low level retries have remained constant since the very first MOST system designs, and have been re-used on subsequent systems. For parameters such as the network services cycle time, only simple guidance might be given to the module suppliers, such as the master node should be faster than the slave nodes. Optimising these parameters could be achieved by practical experimentation. Clearly this could become a long and expensive process, as it would require support from several module suppliers to produce special prototype software and hardware.

The application of the model is to examine the network initialisation strategy and predict the effects of changing parameters, e.g. retry wait times, over a wide variety of initialisation scenarios. A specific question of interest to the technical specialist was; although forbidden by the MOST standard, could varying the low level retry spacing across the network improve performance in a system where only the master node is able to send broadcast messages, and group cast messages are not used?

The objectives are to determine and specify requirements for the MOST system to guarantee predictable initialisation behaviour, and understand the effects of increasing customer feature on the MOST system in advance of building prototypes.

5.4.5 MOST Control Channel Model Implementation

The requirement for the model was to have representative levels of MOST control channel traffic in terms of source, destination and relative timing of messages during the initialisation phase and representative behaviour of the channel for factors such as timing, bandwidth and arbitration.

Based on the generic model previously described, the MOST system was modelled as a series of interlinked state machines. The model was constructed using Matlab Simulink and Stateflow, a top level view is shown in Figure 5.11. The model of a 9 node system was constructed using an existing system as a reference for the communication patterns and MOST specifications to determine underlying behaviours.

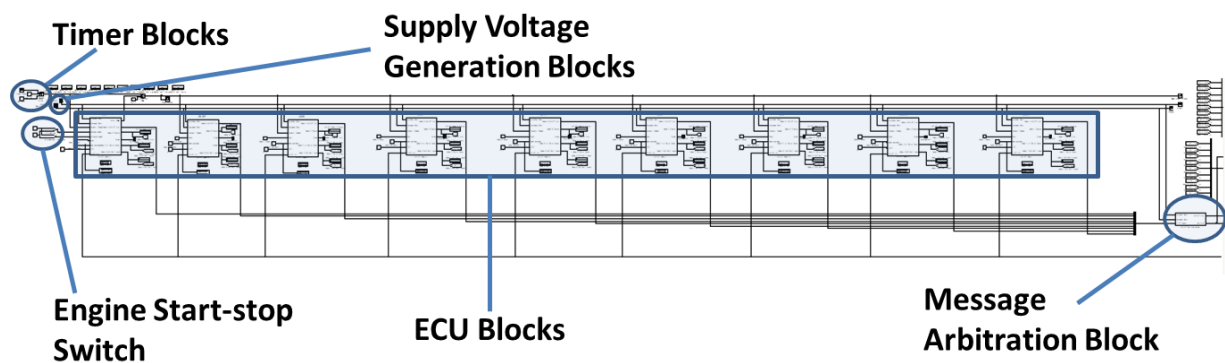


Figure 5.11- Top level view of MOST initialisation model

Figure 5.12 illustrates the top level inputs and outputs of a node. These are similar to the generic models with the following MOST specific additions. The node position and number of nodes allows the ring configuration to be changed. The MOST Bus input is a signal bus from the arbitration node providing: which node has won arbitration transmitted message, the content of the transmitted message, whether messages have been acknowledged, and whether the ring is complete. The MOST Bus output is a signal bus to the arbitration node containing: message transmit requests, content and priority, received message acknowledgement and a binary signal to indicate when the node's transmitting light output is on. The node status output gives an enumerated value corresponding to its stage of initialisation. The node also outputs its current number of low level retries and the number of messages in its send buffer.

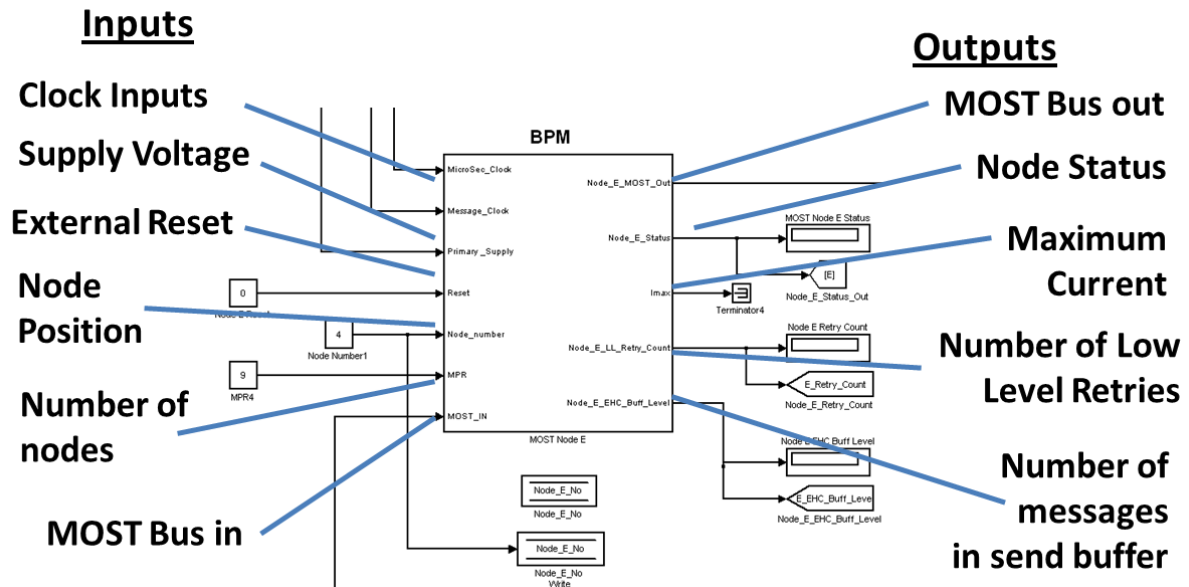


Figure 5.12- Enlarged view of node

Further inputs which were previously identified as impacting initialisation timing and set as user definable parameters within each node are; the minimum operating voltage, the ECU initialisation time, the network interface wakeup time from detecting light at its input, low-level retry interval, the number of low-level retries, the high level retry interval and number, the netservices cycle time, the number of vehicle status requests the node makes during initialisation and the buffer size of the application message handler.

Figure 5.13 shows the high level structure of a node which has similar structure to the generic model with states for unpowered, sleep, initialising and running. States have been added for ring restart to capture behaviour during loss of light around the fibre optic ring, either during shut-down or a fault, and the initialisation has been split into the initialisation of the micro-processor and of the network interface. The major change is in the running state where the key part of the modelling was to determine which aspects of the control channel communications to abstract and how to model them. An approach of only modelling at particular layers in the OSI Reference Model was not possible as the behaviour is a factor of

all layers, so the model included layers used in MOST. These are known as the application layer, application message handler layer, netservices layer and network interface controller and were modelled with appropriate abstractions at each layer which will subsequently be described.

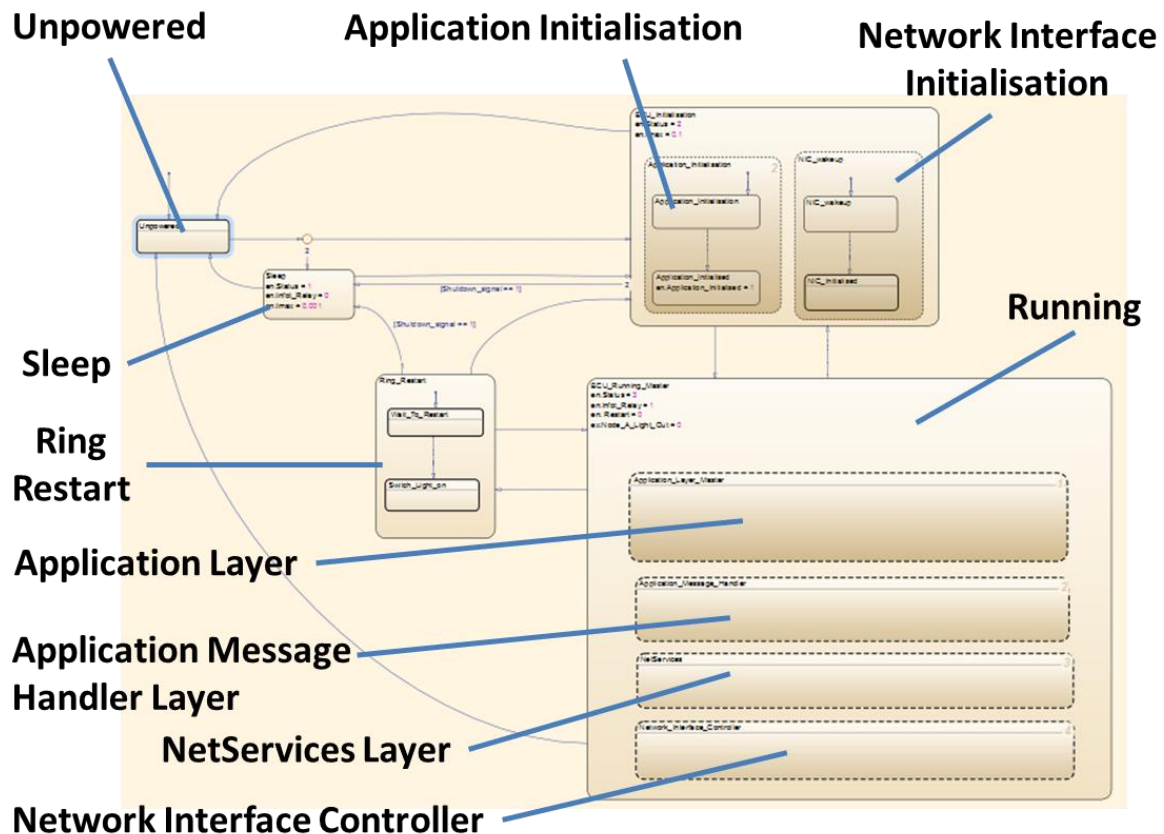


Figure 5.13 - Overall structure of a node in the model

Application Layer

The application layer controls the high level behaviours and implements the initialisation strategy. The top layer of the application layer block is shown in Figure 5.14 illustrating the key stages of the initialisation process which reflects the specified initialisation strategy for the target system. On completion of hardware initialisation the node

waits for the network to become active and then the master sets a timer to allow all participants to join before checking which functions are present on the individual nodes. On receiving a valid configuration the master broadcasts a configuration status OK message which triggers the individual nodes to request via a “Central Registry Get” message either for the full central registry of functions and their location or a portion relating to functions of interest. At this point the slave nodes also request the vehicle configuration from the master and on completion of this the master broadcasts a message to the nodes to commence set-up. This consists of the nodes requesting notification of the status of other functions and is complete on receipt of responses at which time normal running can take place.

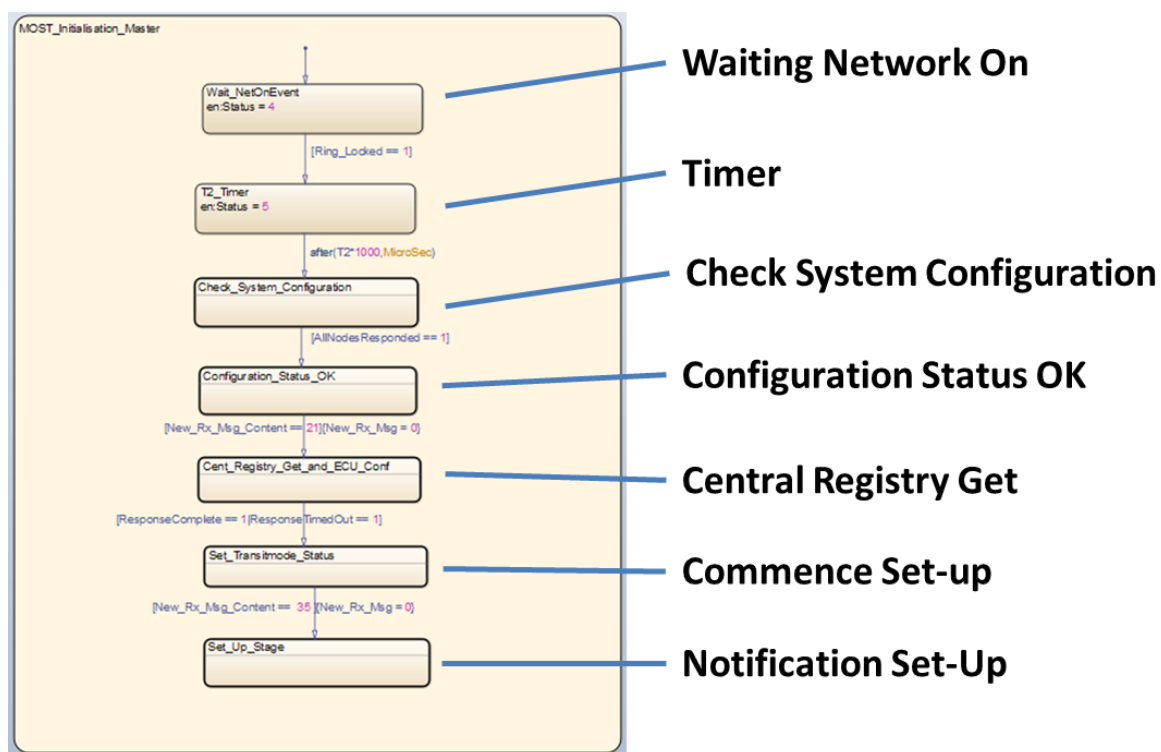


Figure 5.14 Application layer initialisation states in master node

The major abstractions in the application layer were in abstracting the messages into generic classes and simplifying the interaction at application level to generic types or patterns

where the message sent dictated particular response behaviour by the receiver. The actual data was not required merely just enough information to trigger representative message traffic in terms of timing, number, priority, source and destination of messages. Table 5.2 shows part of the message abstraction table which gives the enumerated value of the message types.

<i>Value of Message Contents Variable</i>	<i>Corresponding MOST Message</i>	<i>Comments</i>
0	No message	Default value when no message is transmitted
1 - 14	Reserved	Allows plotting on same graph as Tx/Rx Nodes
15	FBlockIDs.Get	
16	FBlockIDs.Status	
17	FBlockIDs.Status Segmented message not last message	
18	FBlockIDs.Status Segmented message last message	
19		
20		
21	Configuration.Status.ConfigurationControl Ok	Sent when valid central registry built
22	Configuration.Status.ConfigurationControl Changed	
23	Configuration.Status.ConfigurationControl Not Ok	
24	CentralRegistry.Get	
25	CentralRegistry.Get All	
26	CentralRegistry.Status	
27	CentralRegistry.Status Segmented message not last message	
28	CentralRegistry.Status Segmented message last message	
29		
30	Vehicle Status Request (Fblock 0xF5) not last request	ECU Confirmation stage - Each ECU requests vehicle config status from Master & checks (some nodes may have mutiple requests)
31	Vehicle Status Request (Fblock 0xF5) last request	ECU Confirmation stage - Each ECU requests vehicle config status from Master & checks (some nodes may have mutiple requests)
32	Vehicle Status Response (Fblock 0xF5)	ECU Confirmation stage - Response from Master for Vehicle status

Table 5.2 MOST modelling message abstraction table

Generic patterns of interactions during initialisation were observed from specifications and analysis of traces. Two examples of such patterns are shown. Figure 5.15 shows the state machine for behaviours in a slave node during check system configuration phase of initialisation. The node waits for an incoming message type corresponding to an “Fblock request” from the master for it to declare its functions. In response the node creates a new “Fblock Status” request and calls the message handler layer to transmit it. The System check is complete when message handler confirms the message has been successfully transmitted.

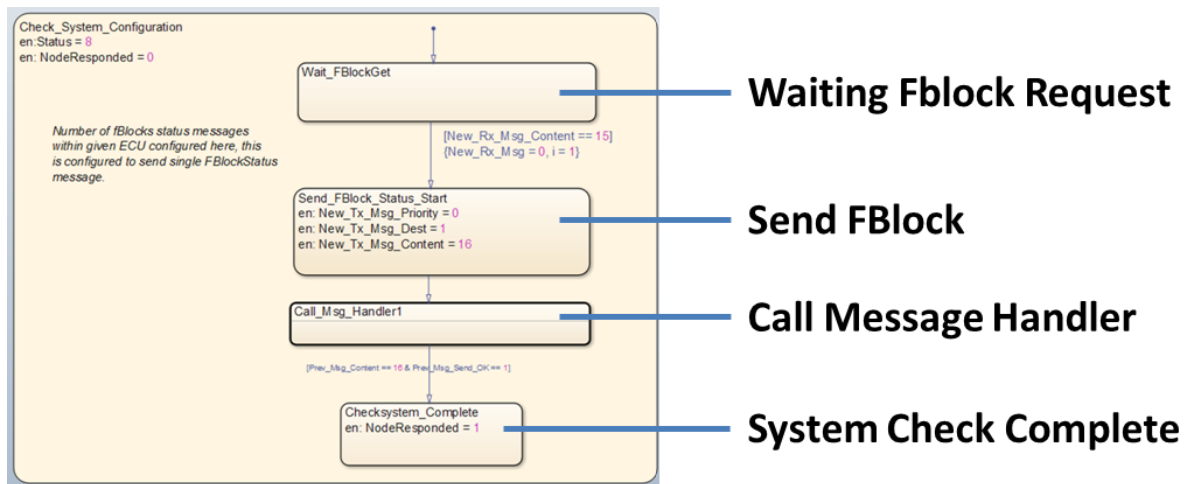


Figure 5.15 Slave node application layer behaviour during check system configuration

Figure 5.16 shows the state machine for behaviour in a slave node during notification set-up phase of initialisation. The node sends a series of notification requests to appropriate nodes using the message handler. The set of notification requests and appropriate responses for each node was derived through analysis of actual system behaviour. These were categorised into different types based on the response from the receiving node. One simplification used in the model was to create globally accessible variables for the location of the nodes in the ring to determine the location nodes should set notification on. This facilitated the abstraction of data from the messaging while maintaining fidelity of message traffic. Notification setting has a high level retry strategy triggered by a message send failure at lower level whereby it will attempt to resend after an interval, typically 500ms.

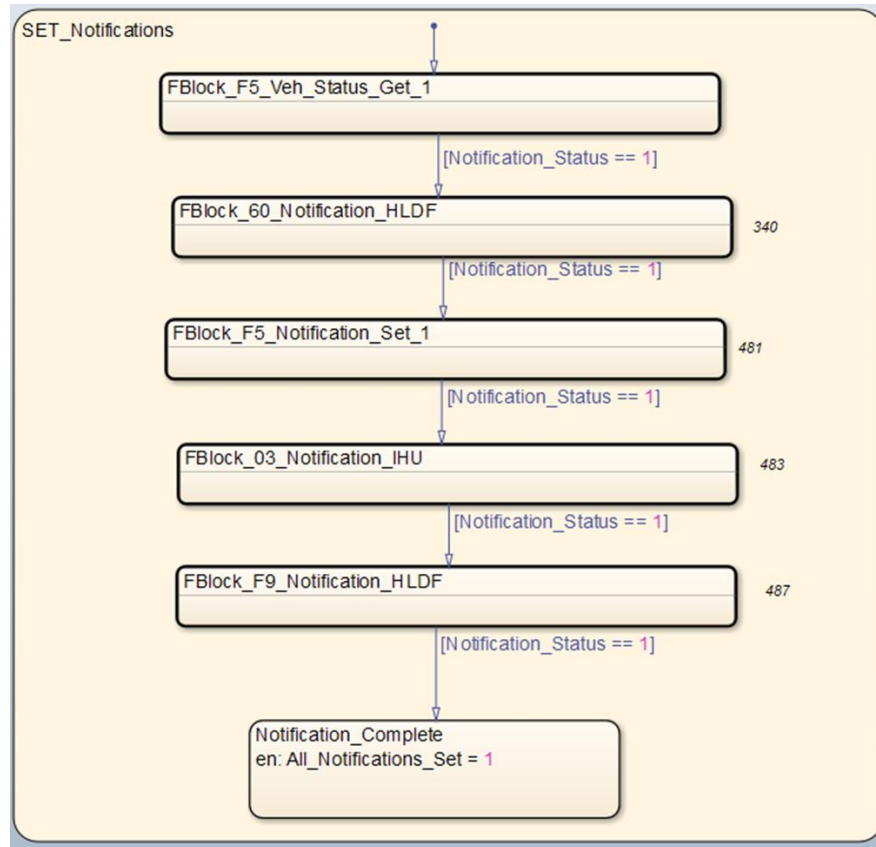


Figure 5.16 Slave node application layer behaviour during notification phase of initialisation

Message Handler Layer

The message handler layer interfaces between application and netservices layers. Figure 5.17 shows the top level structure of this model block illustrating the main functions, which are performed in parallel. The *outgoing message handler* takes send requests from the application level and queues them in priority for the netservices layer. Similarly the *incoming message handler* takes incoming messages from the netservices and queues them to be responded to. The other functions deal with routine responses and consumption of messages which are not part of the node's own initialisation. *Receive FBlock* status consumes the FBlock status messages from nodes where notification has been set. The *notification handler* deals with notification requests from other nodes and generates appropriate status messages

based on the type of notification request. The *Receive vehicle status* function consumes vehicle status messages, other than being received they have no other impact on the initialisation behaviour. *Receive new message processing acknowledgement* is a response to particular types of notification requests where the status information requested is not immediately available and the response is to generate a periodic message confirming the response is still being processed.

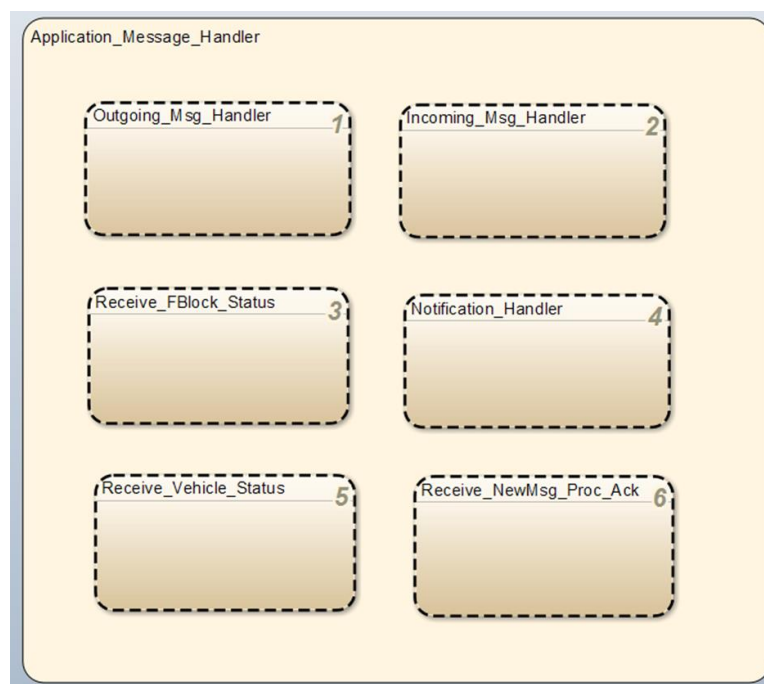


Figure 5.17 Message handler layer functions

Netservices Layer

The Netservices layer is shown in Figure 5.18. The *Call Netservices* block periodically executes the netservices register interface according at an interval defined by a user settable parameter. The *netservices register interface* receives the next message in the network interface controller's incoming message register and loads the next outgoing message to its

outgoing message register and reads the message send result register, reporting back if the previous message has been successfully sent or failed to send.

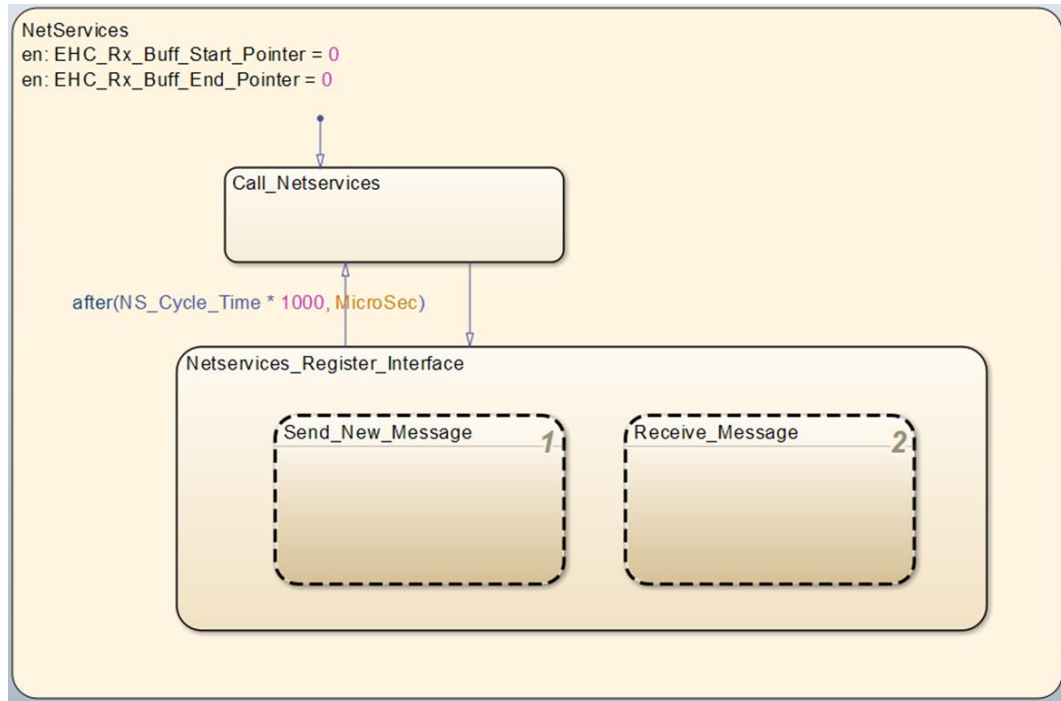


Figure 5.18 NetServices layer functions

Network Interface Controller Layer

The top level functions in the network interface controller layer are shown in Figure 5.19. The *message transmission* function takes an outgoing message, assigns a priority to it according to the message type and the number of times it has failed to win in the arbitration process and sends this to the central arbitration block which is discussed subsequently, via the outgoing message signal bus. If the message is sent but failed to be received it implements the low-level retry strategy according to the set parameters. If still the message fails it records message send failure. The *message reception* function detects if the node is the destination for a transmitted message on the incoming message signal bus. If the received message register is clear it loads the message and sets an acknowledge (ACK) flag, else if the

register still has a message then it set a message not acknowledged (NAK) flag. The values of these flags are sent to the arbitration block via the output signal bus. The final functions relate to the light input and output of the optical physical layer which is abstracted to a Boolean value on the message input and output busses. When the *Set_Node_No_Light_On* function detects the value of the output from the previous node changing from Off to On it sets the value of the lightoutput for its node to On. When the *Monitor_Light_In* function detects the ring on value from the arbitration node it enables message receipt, if the value changes to ring_off it causes a state change from running to ring-restart.

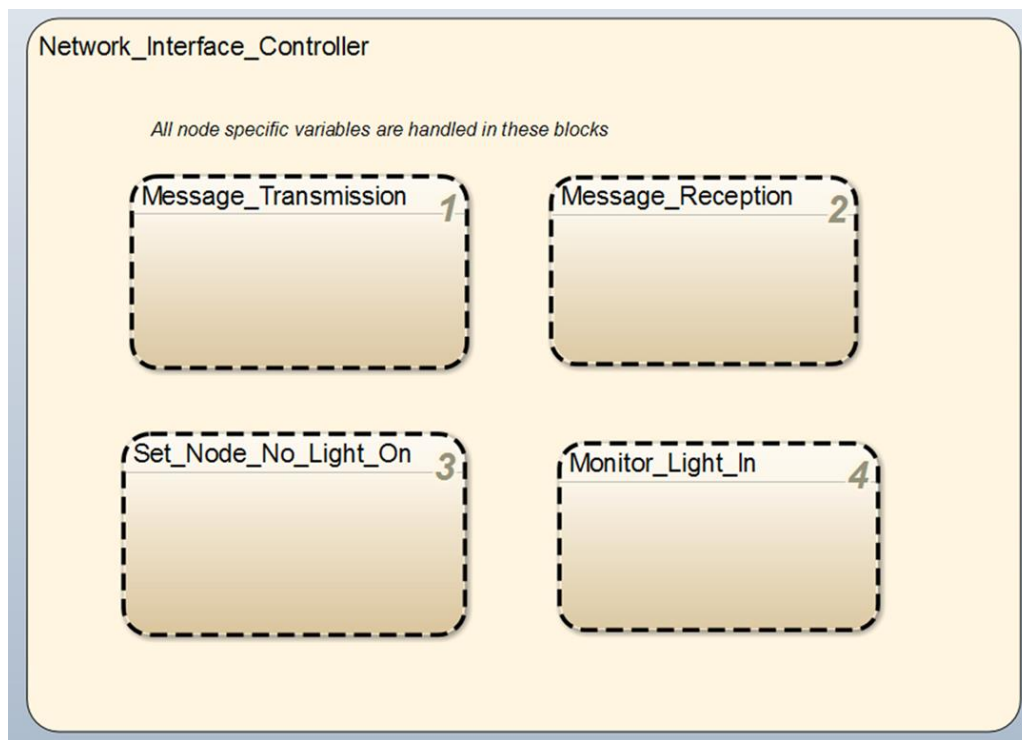


Figure 5.19 Network interface controller block in nodes

A major simplification for the model which reduced the implementation effort while retaining the same communication behaviour, was to model the message arbitration as a centralised process whereas in practise it is a distributed process. Figure 5.20 shows the top

level of the arbitration block which takes the bus access requests from all nodes, determines which has the highest priority, makes the associated message content the content of the transmitted message on the block's output and sets a flag to indicate to the nodes that there is a new message. The arbitration block also sends a message to indicate the arbitration winning block and the result of whether the message is acknowledged or not acknowledged by recipient nodes. A further function is to confirm whether full ring is running by monitoring the light-output of all nodes and output this to the nodes to confirm communication is possible.

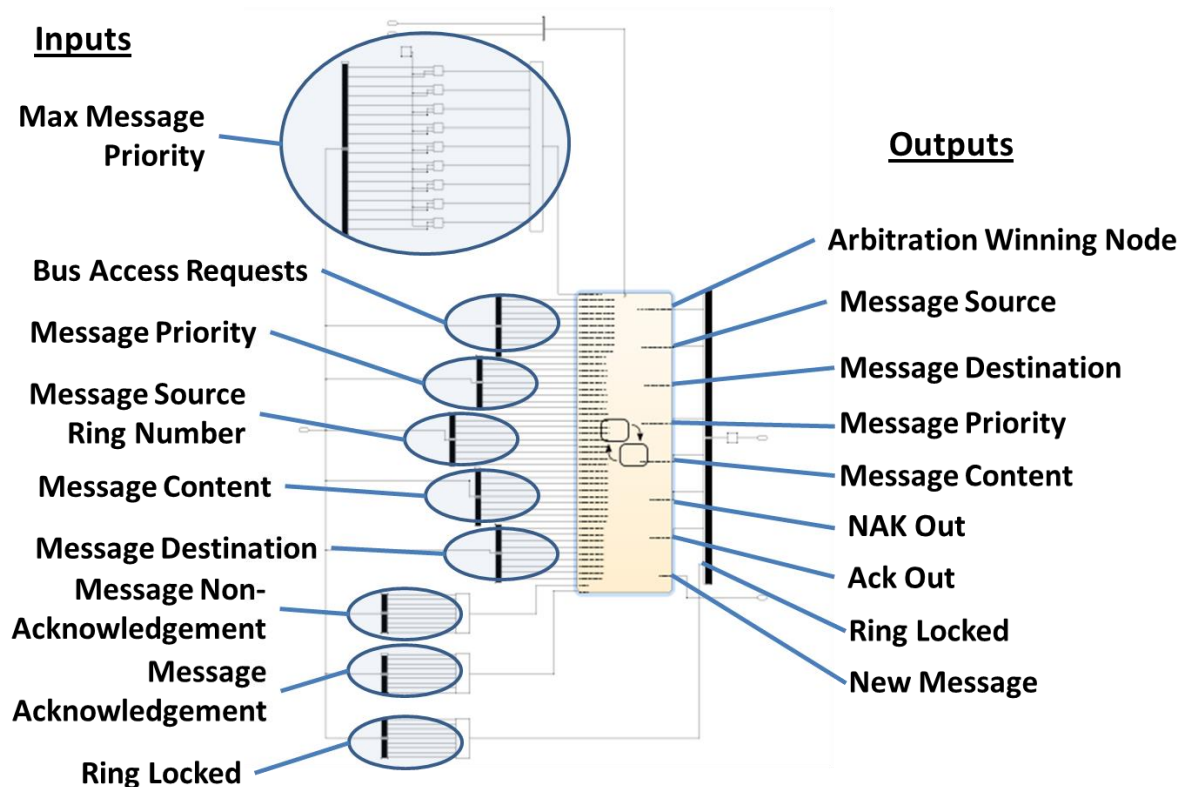


Figure 5.20 MOST Control Channel arbitration block

Finally the model was instrumented with the primary output measurements being: the source, destination and contents of the transmitted messages, the initialisation status of each

node against enumerated stages, the number of low level and high-level retries, the number of not acknowledged messages (NAKs) and the utilization of the internal buffer in the nodes.

5.4.6 Results from Infotainment Robustness Model

Correlation Testing

After debugging, the model was tested for correlation against results recorded from an actual system. Figure 5.21 illustrates part of the message traces obtained from the model against the physical system. Analysis of this and the message traces showed very similar patterns in the communications including known bottlenecks at critical times in the initialisation e.g. when all slave nodes are simultaneously requesting the central registry from the master node.

Model										Physical System										
	Node	Node	Node	Node	Node	Node	Node	Node	Node		Node	Node	Node	Node	Node	Node	Node	Node	Node	
Disassembly	1	2	3	4	5	6	7	8	9	Disassembly	1	2	3	4	5	6	7	8	9	
FBlock/Ds.Get	Source	R								00.NetBlock.01.FBlock/Ds.Get	Source	R								
FBlock/Ds.Status Segmented message	R	Source								00.NetBlock.01.FBlock/Ds.Status SEG/01.R	Source									
FBlock/Ds.Status Segmented message	R	Source								00.NetBlock.01.FBlock/Ds.Status SEG/01.R	Source									
FBlock/Ds.Get			R							00.NetBlock.01.FBlock/Ds.Status.06.8A.1.R	Source	Source								
FBlock/Ds.Status	R		Source							00.NetBlock.02.FBlock/Ds.Get	Source									
FBlock/Ds.Get	Source			R						00.NetBlock.02.FBlock/Ds.Status.31.01.0.R		Source								
FBlock/Ds.Status	R			Source						00.NetBlock.03.FBlock/Ds.Get	Source			R						
FBlock/Ds.Get	Source				R					00.NetBlock.03.FBlock/Ds.Status.91.01.0.R			Source							
FBlock/Ds.Status Segmented message	R			Source						00.NetBlock.04.FBlock/Ds.Get	Source			R						
FBlock/Ds.Status Segmented message	R			Source						00.NetBlock.04.FBlock/Ds.Status.40.02.0.R	Source		Source							
FBlock/Ds.Get	Source					R				00.NetBlock.05.FBlock/Ds.Get	Source				R					
FBlock/Ds.Status	R			Source		Source				00.NetBlock.06.FBlock/Ds.Get	Source					R				
FBlock/Ds.Get	Source						R			00.NetBlock.06.FBlock/Ds.Status SEG/00.R						Source				
FBlock/Ds.Status	R						Source			00.NetBlock.06.FBlock/Ds.Status SEG/01.R						Source				
FBlock/Ds.Get	Source							Source		00.NetBlock.06.FBlock/Ds.Status.50.01.5.R						Source				
FBlock/Ds.Status	R							Source		00.NetBlock.06.FBlock/Ds.Status.50.01.5.R						Source				
FBlock/Ds.Get	Source								R	00.NetBlock.07.FBlock/Ds.Get	Source							R		
FBlock/Ds.Status	R								Source	00.NetBlock.07.FBlock/Ds.Status SEG/00.R							Source			
Configuration.Status.ConfigurationComplete	Source	B	B	B	B	B	B	B	B	00.NetBlock.07.FBlock/Ds.Status SEG/01.R							Source			
CentralRegistry.Get All	Nak	Source								00.NetBlock.07.FBlock/Ds.Status.23.02.0.R							Source			
CentralRegistry.Get	Nak			Source						00.NetBlock.08.FBlock/Ds.Get	Source								R	
CentralRegistry.Get	Nak					Source				00.NetBlock.08.FBlock/Ds.Status.22.01.0.R								Source		
CentralRegistry.Get	Nak						Source			00.NetworkMaster.01.Configuration.Status	Source	B	B	B	B	B	B	B	B	
CentralRegistry.Get	Nak	Source								00.NetworkMaster.01.CentralRegistry.1	R	Source								
CentralRegistry.Get	Nak			Source						00.NetworkMaster.01.CentralRegistry.1F5	Nak			Source						
CentralRegistry.Get	Nak							Source		00.NetworkMaster.00.CentralRegistry.1F5	Nak						Source			
CentralRegistry.Get	R								Source	00.NetworkMaster.01.CentralRegistry.1	Nak							Source		
CentralRegistry.Get All	Nak	Source								00.NetworkMaster.01.CentralRegistry.1F5	Nak			Source						
CentralRegistry.Get	Nak			Source						00.NetworkMaster.00.CentralRegistry.1F5	R						Source			
CentralRegistry.Get	Nak					Source				00.NetworkMaster.00.CentralRegistry.102	Nak								Source	
CentralRegistry.Get	Nak						Source			00.NetworkMaster.00.CentralRegistry.102	Nak									
CentralRegistry.Get	Nak			Source						00.NetworkMaster.01.CentralRegistry.Status	Source	R		Source						
CentralRegistry.Get	Nak				Source					00.NetworkMaster.01.CentralRegistry.1	Nak							Source		
CentralRegistry.Get	Nak				Source					00.NetworkMaster.01.CentralRegistry.1F5	Nak			Source						
CentralRegistry.Get	Nak					Source				00.NetworkMaster.00.CentralRegistry.102	R								Source	
CentralRegistry.Status	Source							R		00.NetworkMaster.00.CentralRegistry.102	R									
CentralRegistry.Get	Nak						Source			00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get All	Nak	Source								00.NetworkMaster.01.CentralRegistry.1	Nak							Source		
CentralRegistry.Get	Nak			Source						00.NetworkMaster.01.CentralRegistry.1F5	Nak				Source					
CentralRegistry.Get	Nak					Source				00.NetworkMaster.00.CentralRegistry.1F5	Nak				Source					
CentralRegistry.Get	Nak							Source		00.NetworkMaster.00.CentralRegistry.102	R			Source						
CentralRegistry.Get	Nak								Source	00.NetworkMaster.01.CentralRegistry.Status	Source	R							Source	
CentralRegistry.Get	Nak			Source						00.NetworkMaster.01.CentralRegistry.1	R									
CentralRegistry.Get	Nak				Source					00.NetworkMaster.01.CentralRegistry.1F5	Nak			Source						
CentralRegistry.Get	Nak					Source				00.NetworkMaster.00.CentralRegistry.1F5	Nak				Source					
CentralRegistry.Get	Nak						Source			00.NetworkMaster.00.CentralRegistry.1F5	Nak			Source						
CentralRegistry.Get	Nak							Source		00.NetworkMaster.01.CentralRegistry.Status	Source	R			Source					
CentralRegistry.Get	R			Source						00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Status	Source				R					00.NetworkMaster.01.CentralRegistry.1F5	R									
CentralRegistry.Get	Nak			Source						00.NetworkMaster.00.CentralRegistry.1F5	Nak			Source			Source			
CentralRegistry.Get	Nak					Source				00.NetworkMaster.00.CentralRegistry.1F5	Nak									
CentralRegistry.Status	Source									00.NetworkMaster.00.CentralRegistry.1F5	Nak									
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R								
CentralRegistry.Get	Nak									00.NetworkMaster.01.CentralRegistry.Status	Source	R				</				

Figure 5.21 Comparison of message traces obtained from the model and from the physical system

Figure 5.22 illustrates the output from the model of the initialisation status of the nodes against the enumerated stages. Overall the model initialisation time was faster than the actual system; the cause of this is mainly due to there being behaviours in some nodes of the actual system not being to specification (the system was still in development at this stage), causing parts of the initialisation to be repeated and also due to application level delays in the model where actual nodes may have other concurrent application level tasks. This resulted in a greater intensity of communications leading to a higher level of low-level and high-level retries in the model. For the initial work in investigating methods of configuring the system

to reduce initialisation delays, this greater intensity of communication was desirable to enable testing with a degree of stress.

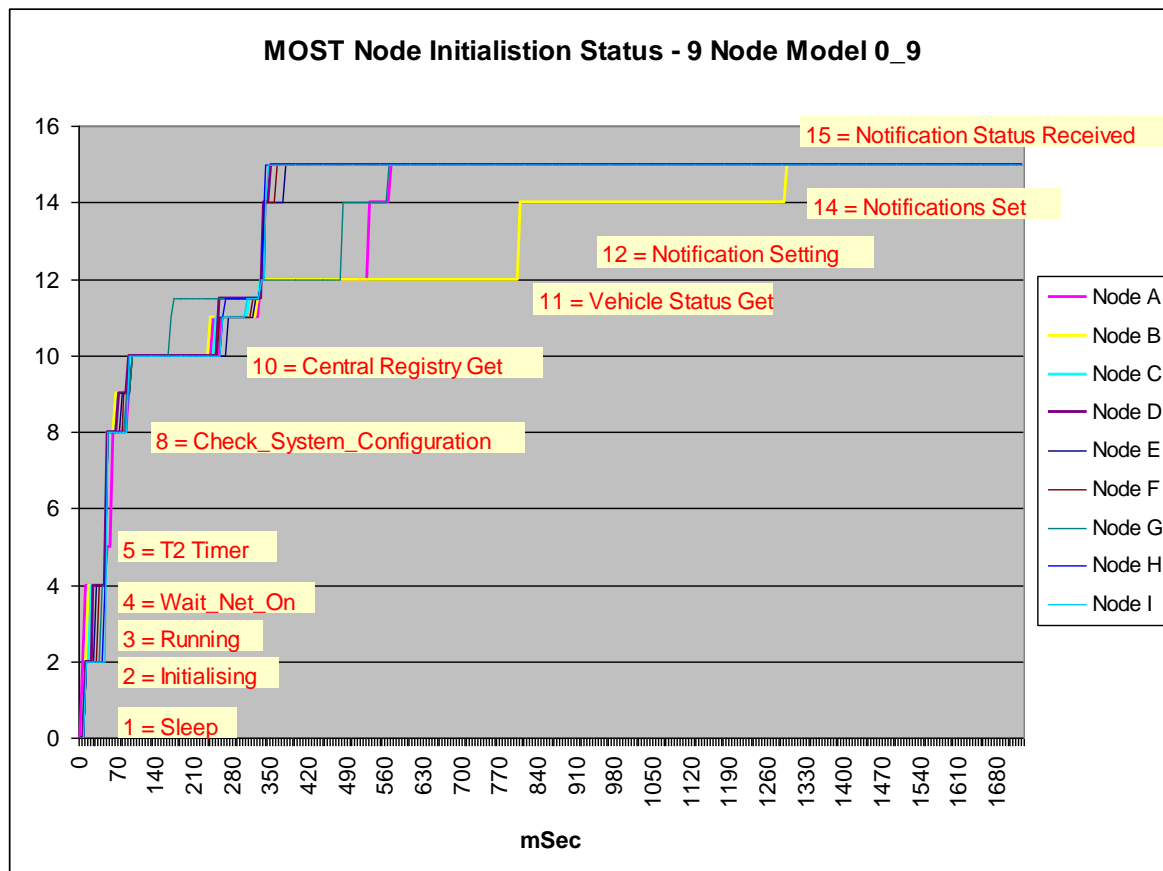


Figure 5.22 Initialization status trace from MOST Control Channel model

The model size grew substantially relative to the generic model with the increase in complexity of the nodes and the number of nodes on the target system. As a result the run time of the simulations increased to becoming around 1000 times slower than real-time on a standard lap-top. However as the period of interest is comparatively short (typically under 10 seconds) this was found to be acceptable.

Initialisation Parameter Robustness Testing

The objective of this testing was to understand the critical parameters to the robustness and consistency of the initialisation process and to be able to investigate alternative strategies. The key parameters to be investigated were NetServices timing (Test 1), Low Level re-try timing (Test 2), Low Level retry number (Test 3) and Ring order (Test 4).

Base Case

For the base case testing of the model the parameter values, shown in Table 5.3, were set to that of the actual specified system. Exceptions to this were factors which added a time delay without interacting with any other factors but were constant in all tests (T2 timer, ECU and network interface initialisation times). These were reduced to minimal values, which significantly reduced the time for each individual simulation run.

Parameter	Specification		Node A	Node B	Node C	Node D	Node E	Node F	Node G	Node H	Node I
	Master	Slave	IHU	HLDF	AUD	CDC	BPM	AUU	RSE	SDARS	IBOC
Net Services Cycle Time	<4mS	Max 4mS	4	4	4	5	4	5	4	4	8
Low Level Retry Time	4	4	4	4	4	4	4	4	4	4	4
Low Level Retry Time No.Blocks (3min)	11	11	11	11	11	11	11	11	11	11	11
Number Low Level Retries	11	11	11	11	11	11	11	11	11	11	11
Central Registry Get High Level Retry Time (mS)		200/500		500	200	200	200	200	500	200	200
Transit mode send High Level Retry Time (mS)	1000		1000								
Vehicle Status High Level Retry Time (mS)		500 x 8		500 x 8	500 x 8	500 x 8	500 x 8	500 x 8	500 x 8	500 x 8	500 x 8
Notification Set High Level Retry Time (mS)	500 x 4	500 x 4	500 x 4	500 x 4	500 x 4	500 x 4	500 x 4	500 x 4	500 x 4	500 x 4	500 x 4
Ring Number			1	2	3	4	5	6	7	8	9
T2 Timer (delay in mS for application initialisation)			10								
Tmax Config OK (mS)			10000								
ECU Init Time (mS)			5	5	5	5	5	5	5	5	5
NIC Wakeup (mS)			1	5	5	5	5	5	5	5	5
EHC Receive Buffer Size			10	10	10	10	10	10	10	10	10

Table 5.3 Base Case parameter values

The results for the base case are shown in Figure 5.23. The initialisation time is less than would be expected for an actual system due to the reduction in constant parameters to reduce simulation time; however the important factor is the relative initialisation times which

this does not affect. The number of not acknowledged messages (NAKs) and re-tries are also deliberately higher than actual due to the lack of delays at application level previously noted.

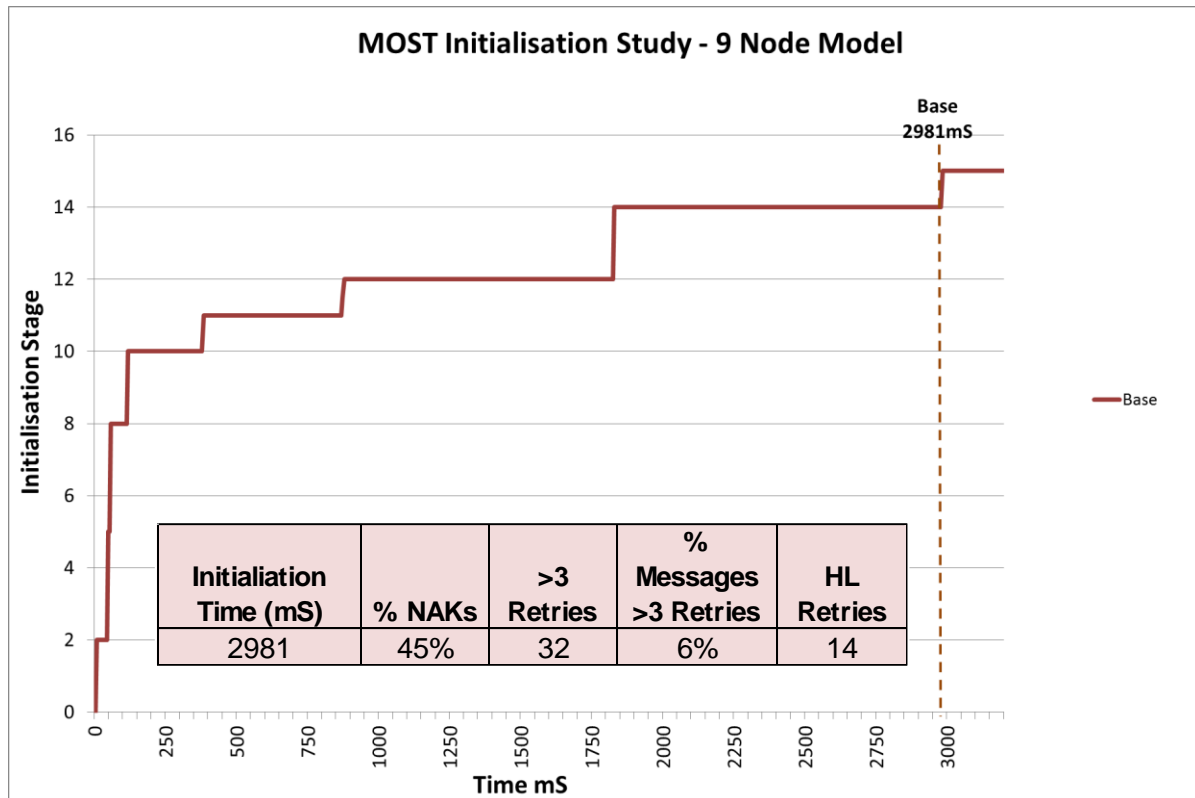


Figure 5.23 Base case results

Effect of NetServices Timing (Test 1)

The first area of investigation was to assess the impact of changing the frequency at which MOST netservices, which takes messages from and to the message buffers of the network interface, is run by the nodes. The test hypothesis was that increasing the frequency of netservices should reduce initialisation time. However to run faster may result in additional hardware costs so it is important to understand where the best gains can be made.

In Test 1a the netservices timing for all nodes was set to 4ms which was a reduction from 5ms in Nodes D and F, and from 8ms in Node I. As shown in Figure 5.24 this did result in a significant decrease in initialisation time to 2143ms. The major gain of over 500ms was

during initialisation stage 11, where vehicle status is requested by all nodes from the master with smaller gains at other stages. A high level retry during stage 11 as a result of a failed message send to the master node by the HLDF (High Line Display Front) which is present in the base case was avoided. Further analysis of the message traces from the model showed that the message sent failure is only narrowly avoided in Test 1a highlighting an emergent property whereby a slight improvement overall communication can make step change improvements in performance.

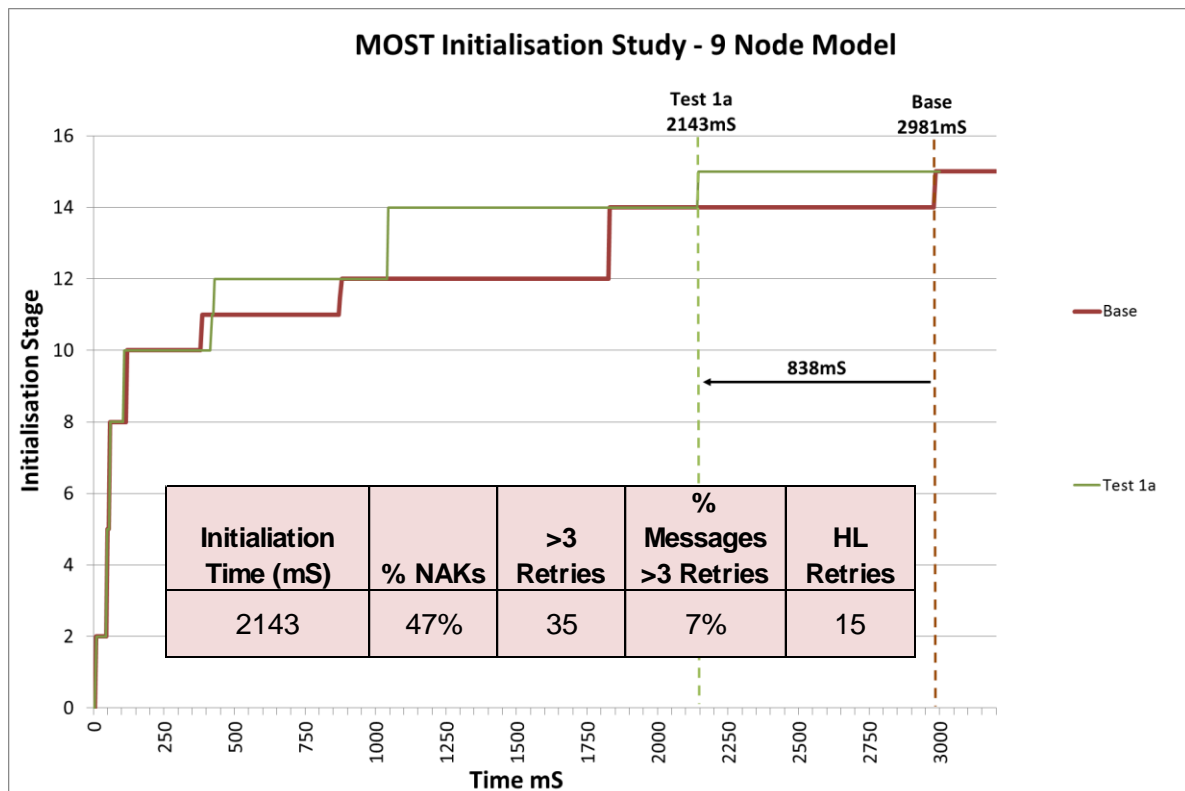


Figure 5.24 Test 1a Net Services for all nodes at 4ms

The next test (1b) was to investigate if increasing the frequency of running netservices on the master node, which has the highest overall number of messages sent and received during initialisation, to 2ms would further decrease initialisation time. The results, shown in Figure 5.25, show that in practise this did not happen. While still being an improvement on

the base case initialisation timing of 478ms it was slower than Test 1a with all nodes at 4 ms Netservices cycle time.

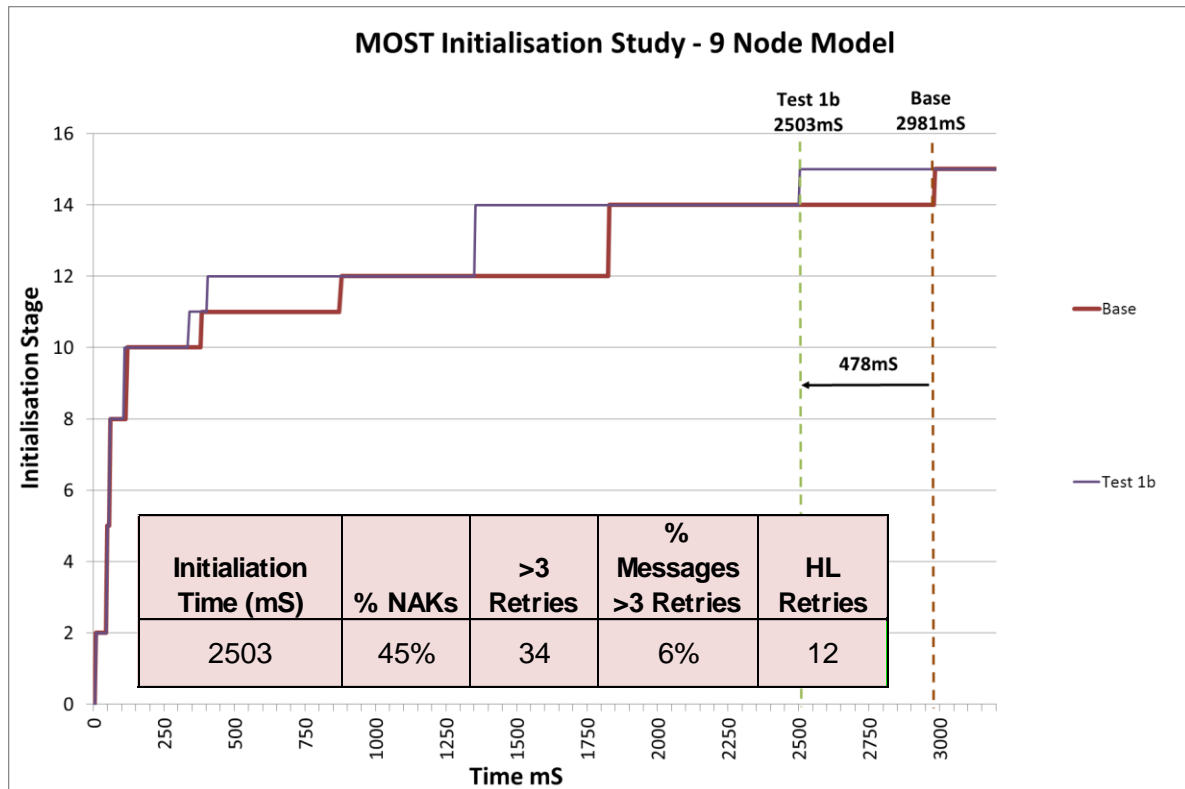


Figure 5.25 Test 1b Master cycle time reduced to 2ms

The key difference was in stage 12 of the initialisation, which is for notification setting, with this taking 950ms for Test 1b against only 615ms in Test 1a. Further analysis of the communication traces revealed that in this stage the node on the critical path was in fact one of the slave nodes, the HLDF (High Line Display Front) which sets the highest number of notifications to be able to display system status to the driver. When set to the same netservices time the HLDF and the master node set notifications in parallel, but when the master set the higher net services cycle time it dominates and causes the HLDF to do a high-level retry. This results in the notifications being set in series with HLDF notifications being

delayed until after the master node has completed resulting in overall longer time for this phase.

The next test (1c) was to see the effect of also reducing the netservices cycle time down to 2ms for the next two nodes with high communication intensity during initialisation. The results, shown in Figure 5.26, show that, while there was an improvement over previous Test with just the master node at 2 ms, it was still not as good as Test 1a with all nodes set to 4ms.

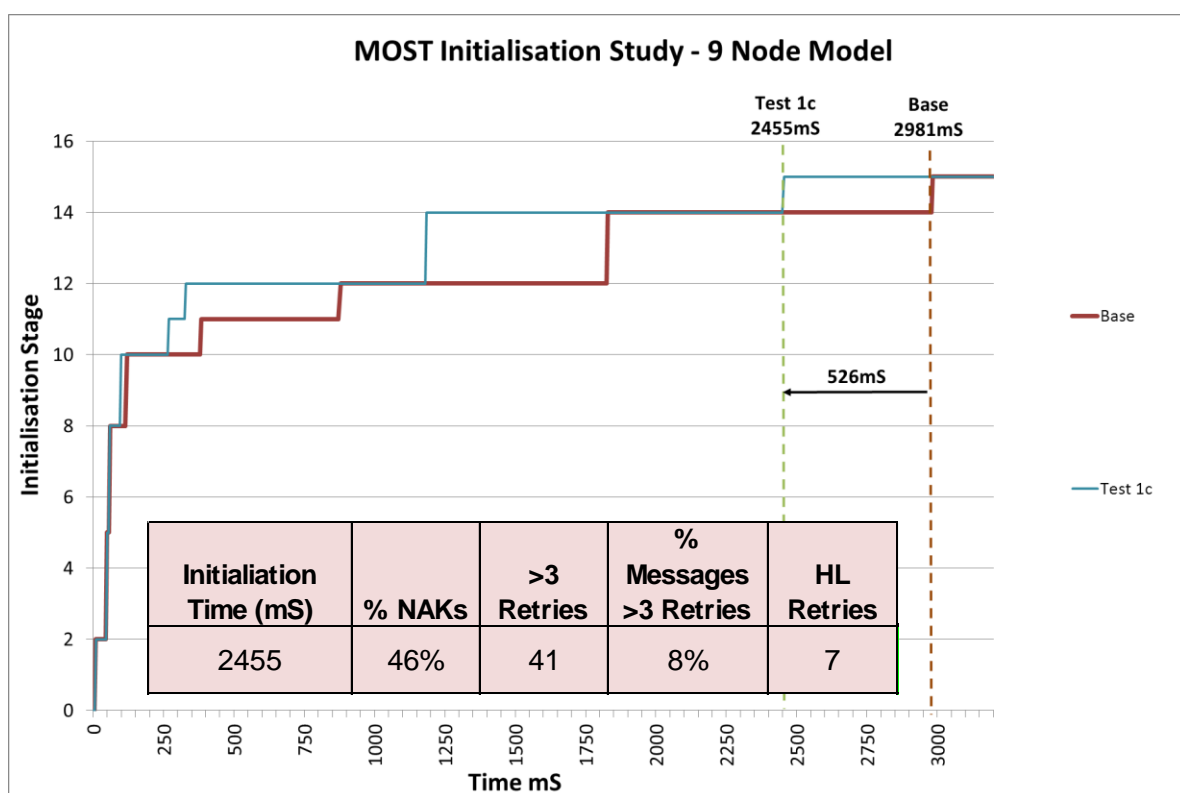


Figure 5.26 Test 1c Key nodes (IHU,HLDF,RSE) cycle time reduced to 2ms

On examination of the communication traces it could be seen that the critical stage was again notification set up with this taking 855ms as opposed to 615ms Test 1a. The cause was that the HLDF made a high level retry due to now competing unsuccessfully with RSE (Rear Seat Entertainment) node and other nodes to get message received by master. Further analysis suggested that in times where many nodes are competing to get message response

from the master node it may be undesirable for them to be synchronised by using the same net services frequency or an exact multiple of this.

For the next test (1d) the net services frequency of remaining nodes was reduced to 3 ms to desynchronise them from the nodes at 2ms. The results of this are shown in Figure 5.27. They show that this de-synchronisation did indeed result in a substantial decrease in initialisation time to 1309ms as a result of the HLDF not requiring a high-level retry during the notification setting stage.

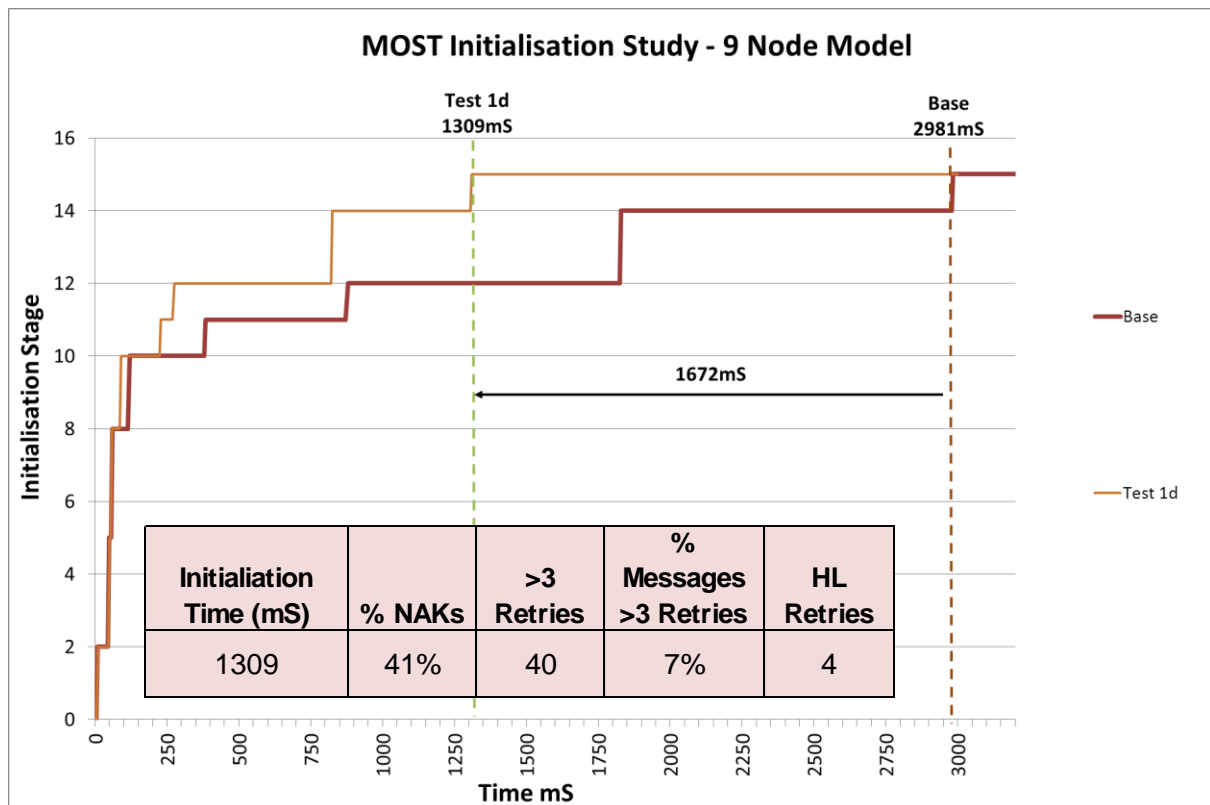


Figure 5.27 Test 1d Key nodes (IHU,HLDF,RSE) cycle time reduced to 2ms, other nodes reduced to 3ms

The next test was to take the de-synchronisation concept further and repeat Test 1d but slightly skewing net services cycle time for each node such that no two nodes were running on

same cycle time. The results shown in Figure 5.28 indicate a slight but not significant benefit over Test 1d for doing this.

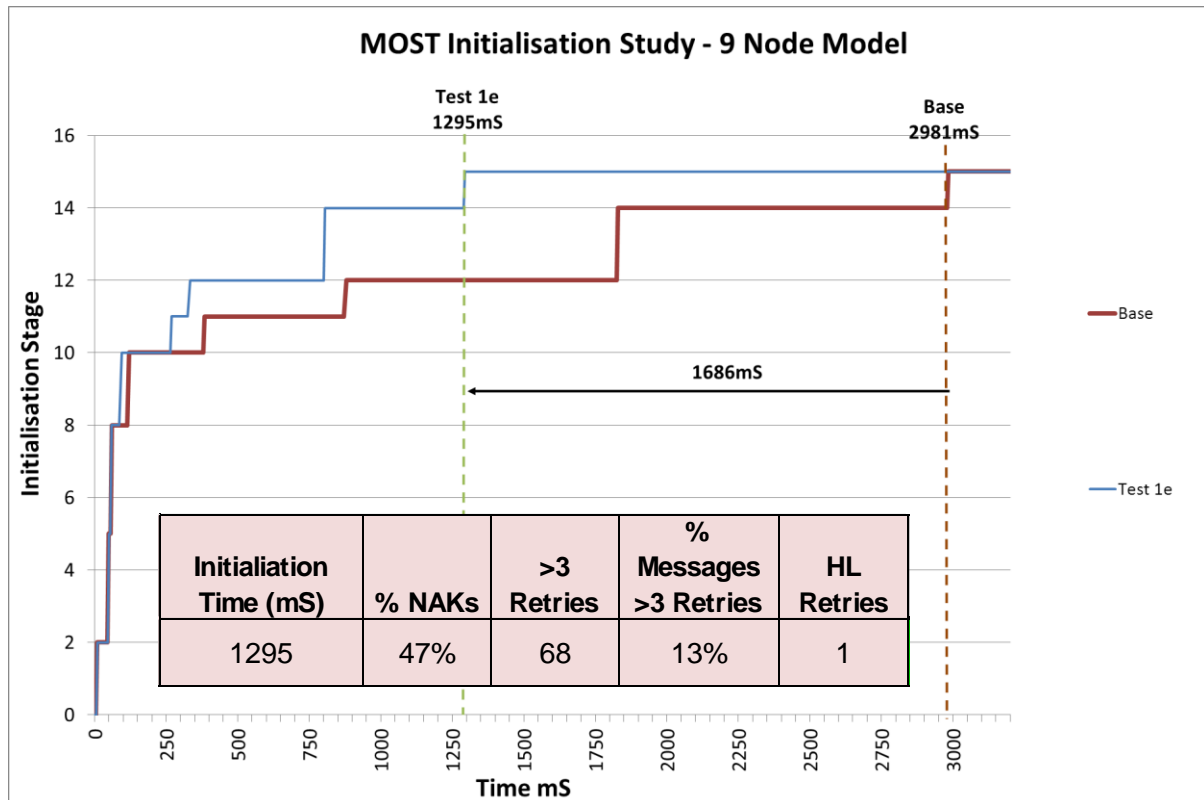


Figure 5.28 Test 1e Key nodes (IHU,HLDF,RSE) cycle time reduced to c.2ms, other nodes c.3ms but not multiples

Low Level Re-try Timing (Test 2)

The next test was to determine if there were benefits in deviating from MOST specification by deliberately introducing a variation in Low Level retry times between nodes to desynchronise retries. This would only be possible if no other node than the master was allowed to send broadcast messages but this is the case in the system under consideration.

The first test (2a) reduced the low level message retry from 4ms to 2ms on the master node, this being the sender of more messages during initialisation than any other node. The expected result was a reduction in initialisation time through the master node winning

message arbitration more often. However the actual result shown in Figure 5.29 was a slight increase of 201ms in initialisation time, this being all in stage 12 of notification setting due to an additional high level retry as a result of slight change in phasing due to master being able to set notifications quicker, with timing of other initialisation stages being unchanged.

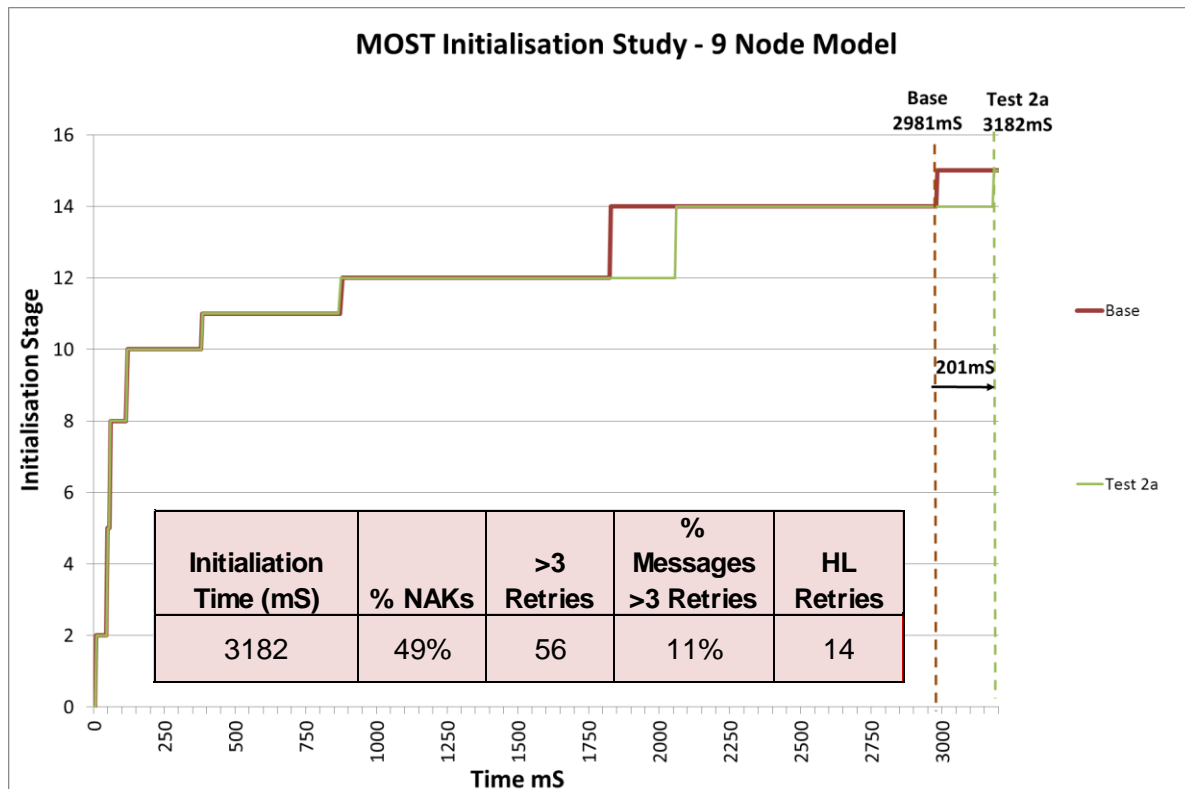


Figure 5.29 Test 2a BASE with master on 2ms Low Level retry

The reason most stages are unaffected is that generally messages *from* the master were being received without retries and the pace of reception was being set by the receiver in phases where there was little other bus contention.

For the next test (2b) a larger variation was tried across all nodes as shown in Table 5.4, the values being allocated according to number of messages sent during initialisation.

	Node A	Node B	Node C	Node D	Node E	Node F	Node G	Node H	Node I
Parameter	IHU	HLDF	AUD	CDC	BPM	AUU	RSE	SDARS	IBOC
Low Level Retry Time	4	4.5	6	7	8	9	5	10	11

Table 5.4 Test 2b variations in low level retry timing

The results of test (2b) are shown in Figure 5.30. This shows a 650ms reduction in initialisation time relative to the base case which has been achieved through avoiding a high level retry during stage 11 of the initialisation. Time for other phases has not been adversely affected showing potential benefits for the strategy.

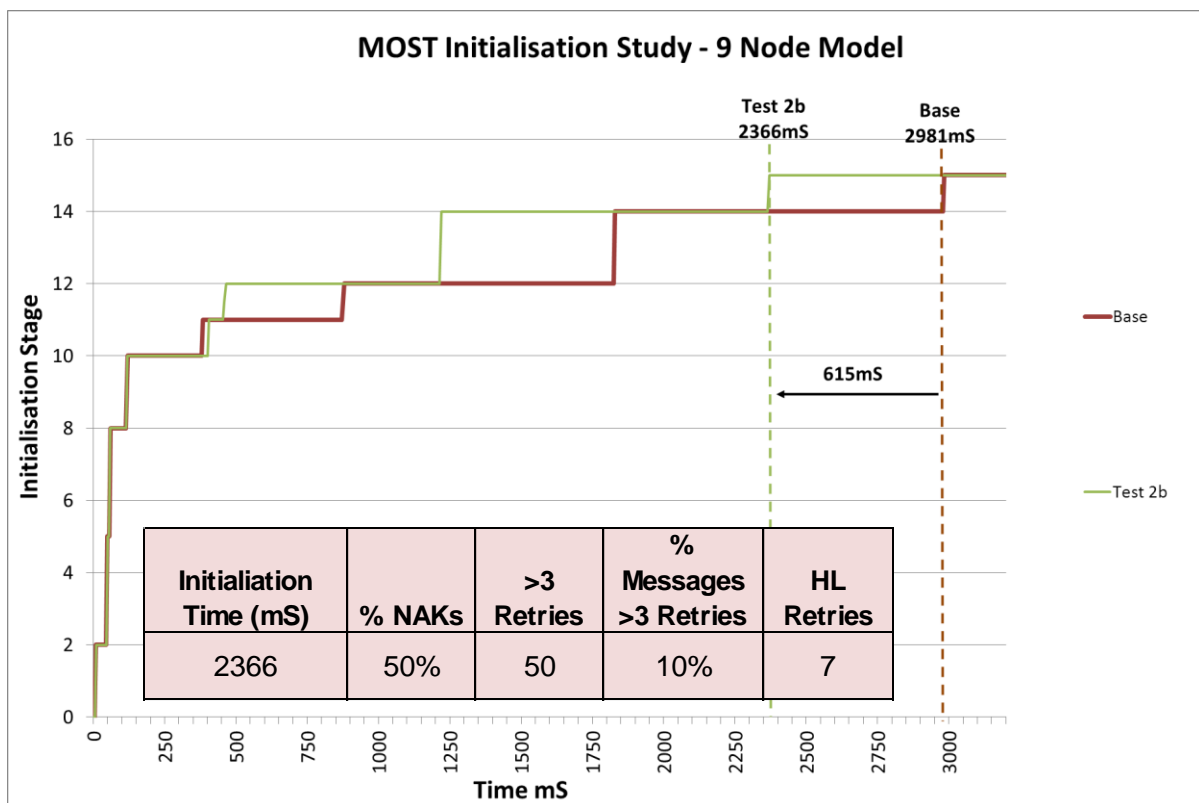


Figure 5.30 Test 2b BASE with all different retry timing - large variation

A subsequent test (2c) was carried out with a much smaller variation in retry timing shown in Table 5.5. The objective of this was to desynchronise the retry timing of the nodes while adding the minimum extra timing. Again these were allocated on the basis of a

number of messages sent during initialisation. Note the variation has to be in discrete steps the numbers allocated increased incrementally by a single step.

	Node A	Node B	Node C	Node D	Node E	Node F	Node G	Node H	Node I
Parameter	IHU	HLDF	AUD	CDC	BPM	AUU	RSE	SDARS	IBOC
Low Level Retry Time	2	2.541	5.082	4.719	4.356	2.904	3.63	3.267	3.993

Table 5.5 Test 2c variations in low level retry timing

The results of Test 2c are shown in Figure 5.31. Again the desynchronisation of retry timing gave a benefit in reducing initialisation time by 448ms, however this time the benefit was achieved in steps 12 and 14 of the initialisation.

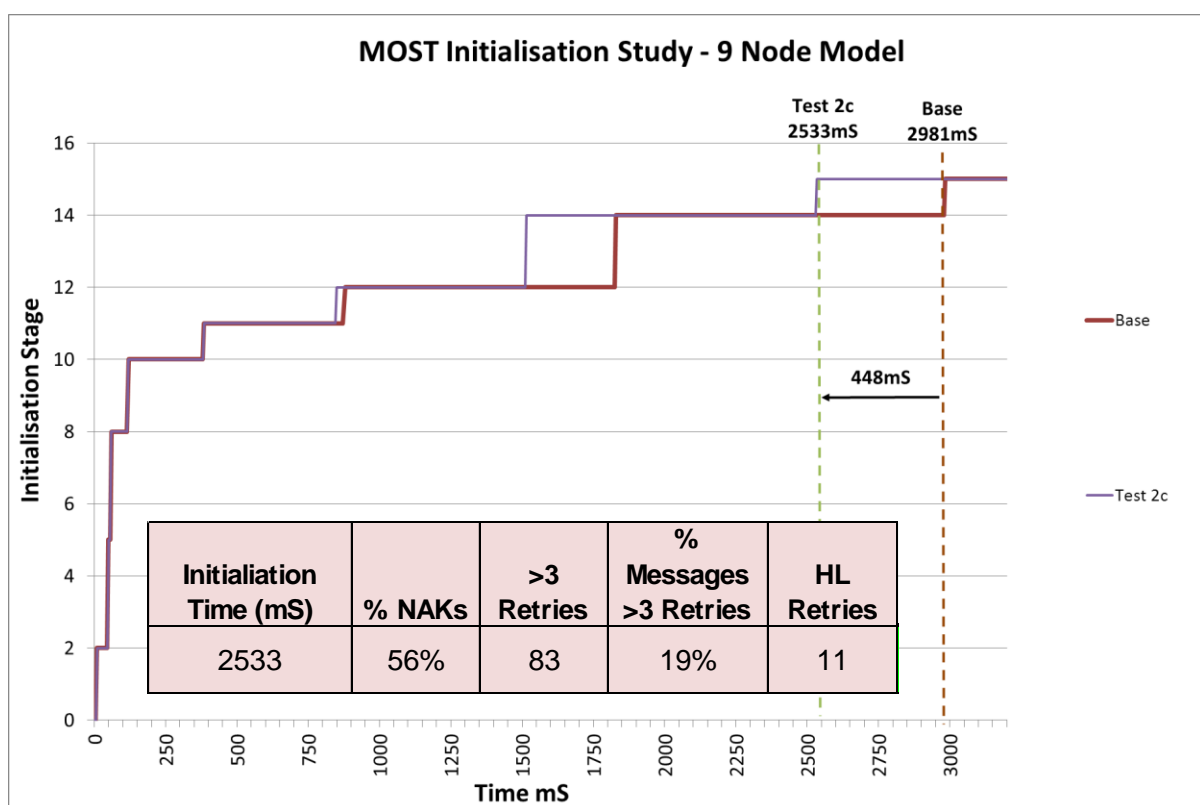


Figure 5.31 Test 2c BASE with all different retry - small variation

Low Level Retry Number (Test 3)

The next set of tests looked at the impact of increasing the number of low level retries with a view to avoiding delays due to high-level retries during initialisation. In the first test (3a) the low level retry number was increased from 11 to 15 on all nodes. Figure 5.32 shows the results which give an initialisation time reduction of 828ms. This was mainly as a result of avoiding a high level retry during stage 11 of the initialisation on the base case but there was also a slight benefit in avoiding a high level retry in stage 12.

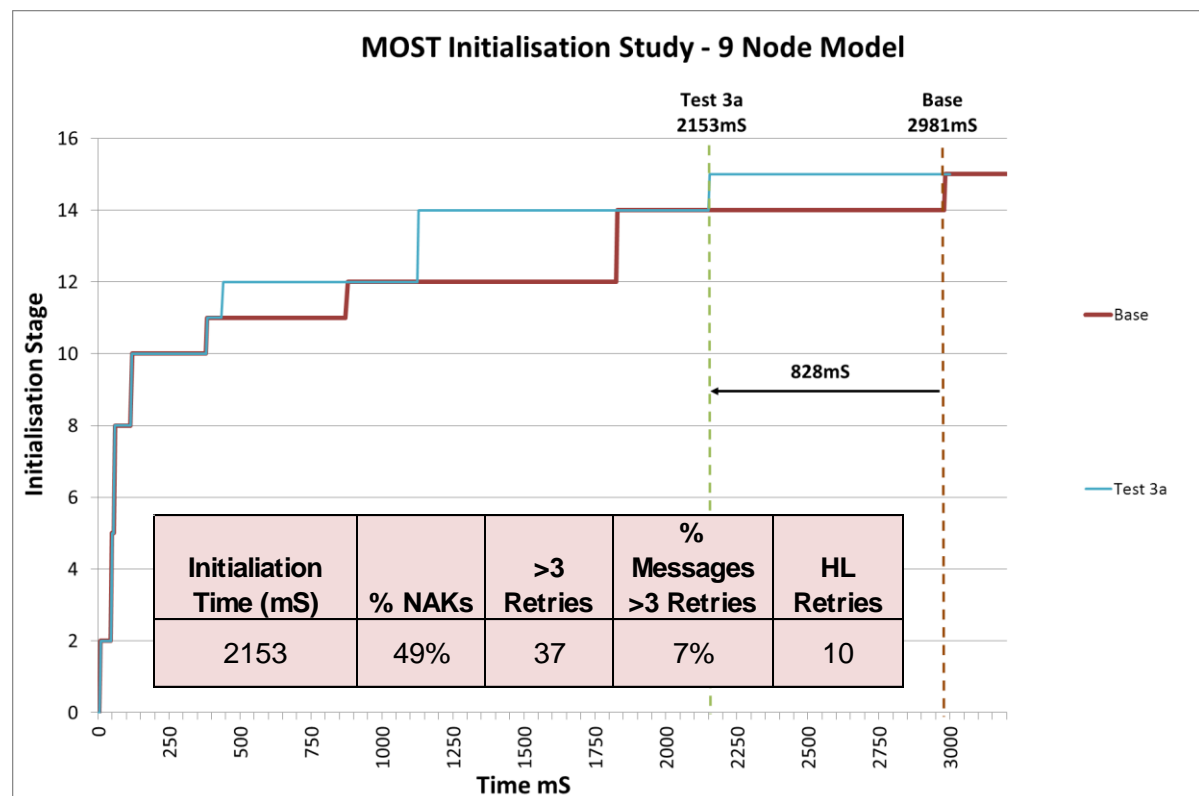


Figure 5.32 Test 3a Base but with low level retry increase to 15 for all nodes

For the next test (3b) a strategy was investigated of only increasing the low level retry to 15 on the nodes which exhibited a high level retry, i.e. had exceeded the maximum number of low level retries, on the base case. The results, shown in Figure 5.33, show a similar level of benefit stage in 11 in avoiding a high level retry but not in stage 12.

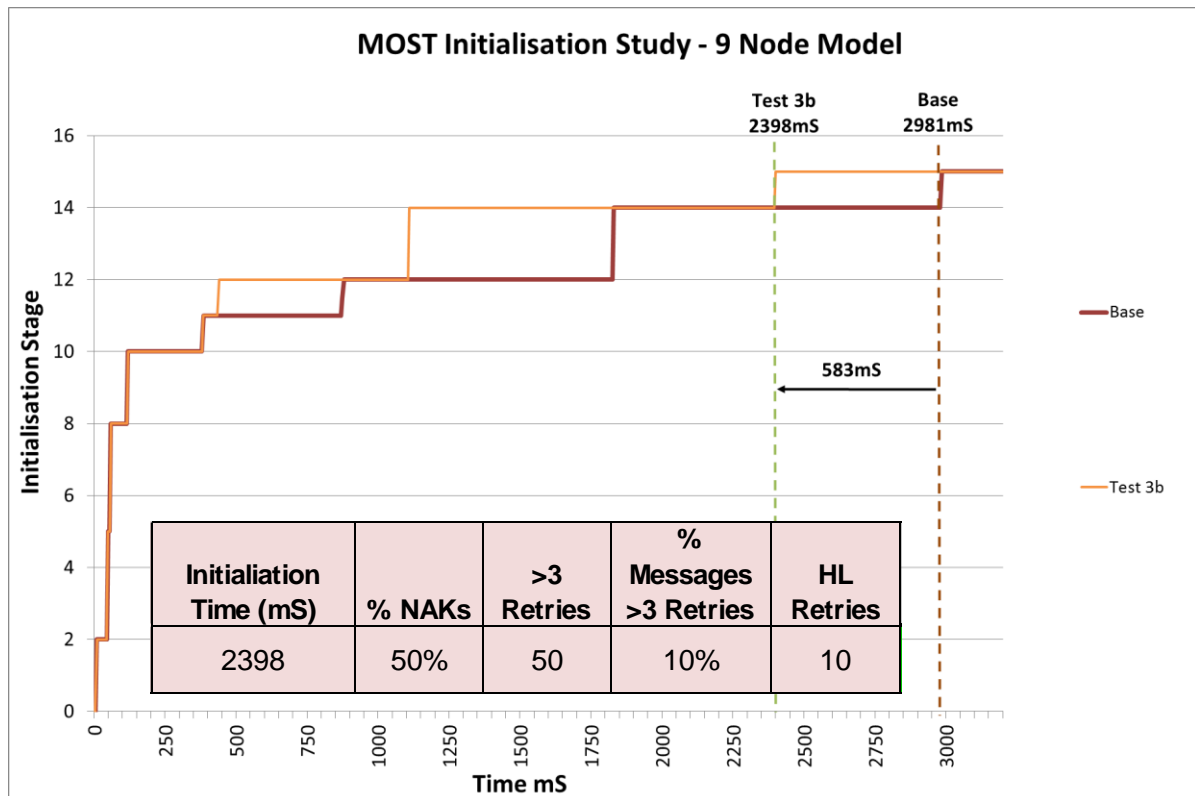


Figure 5.33 Test 3b Differentially increase nodes with 2 HL retries in Base (IHU,HLDF,AUD,AUU,SDARS) to 15

Ring Order (Test 4)

According to MOST specifications ring order is not a factor in message arbitration. However it was hypothesised that if all other factors are equal in the control channel message arbitration then ring order does become the deciding criteria. In Test 4 a node with a heavy utilisation of the control channel particularly in terms of received messages was moved from directly after the master to last in ring order. The results shown in Figure 5.34, show that this slight change was enough to prevent a high level retry which resulted in a reduction in initialisation time. While the high level retry distorted the impact of the effect it shows that ring order can have an effect.

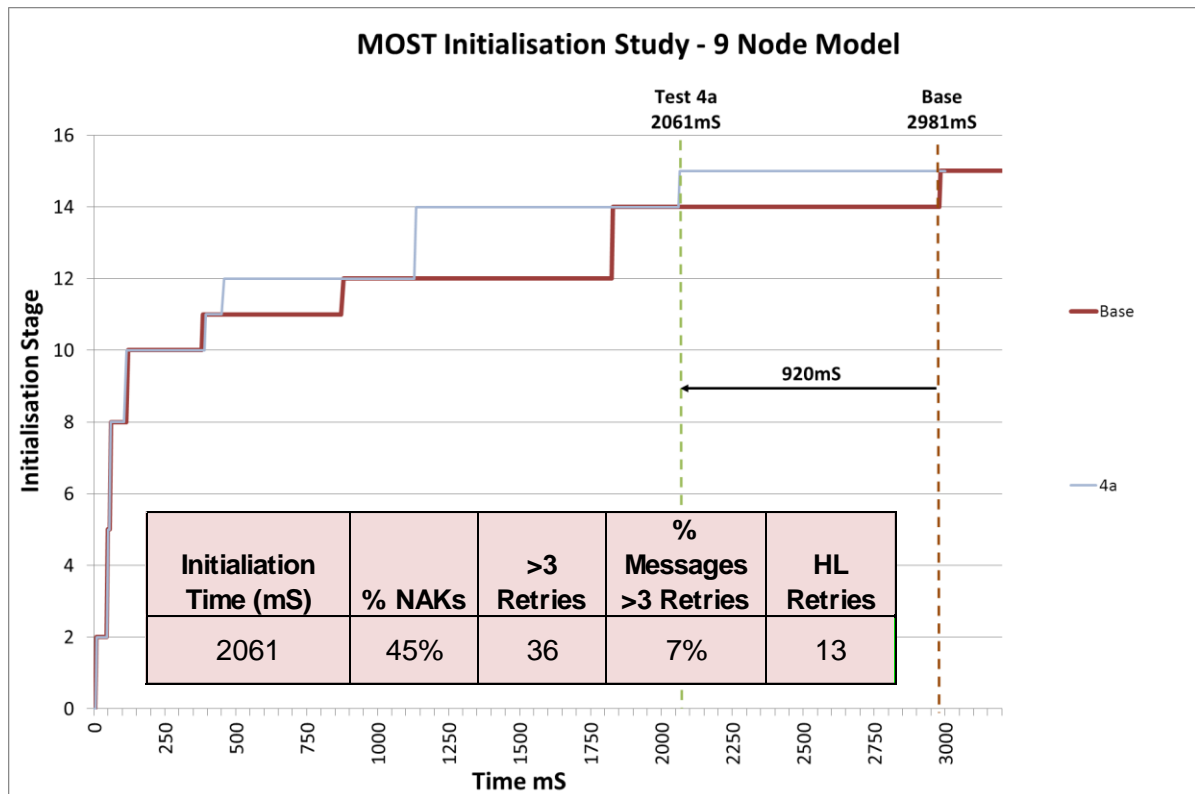


Figure 5.34 Test 4 HLDF moved from 2nd to last in ring order

Overall Comparison

An overall comparison of results from the investigation into changing initialisation parameters is shown in Figure 5.35. This highlights the potential benefits which could be derived from exploring different strategies including some which can be done at no cost such as increasing the number of retries (Testt 3) or changing ring order (Testt 4). The greatest benefits are derived through increasing the frequency of net services (Testt 1) but care needs to be taken to ensure the benefit is on nodes on the critical path and that no adverse effects in other ECUs. Clearly there is scope for further experimentation looking at the effects of combining strategies. An initial test was carried out on two potential hybrid strategies but in these cases the benefits were not additive and the improvement was not better than the lower of the individual strategies they were based on. However what this does show is that the

model can facilitate more complex designs of experiments to look at the effects of interactions of multiple variation parameters.

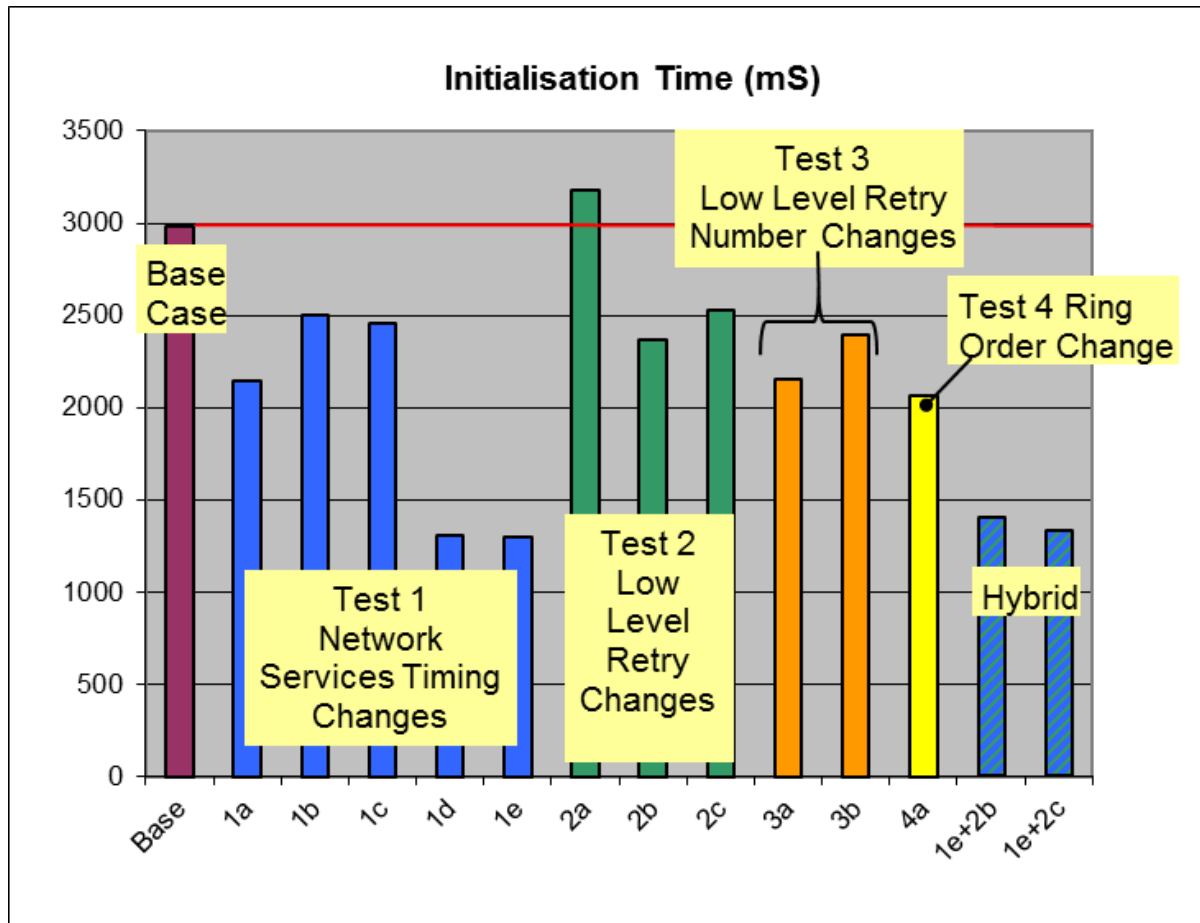


Figure 5.35 Overall test results from 9 node MOST system

A key finding is that due to the non-inherent non-linear properties of the system a small improvement could provide a significant benefit.

5.5 Initial Conclusions on Application of Methods

5.5.1 Infotainment Robustness Case Initial Conclusions

Based on the application of robustness cases to infotainment systems it appears to be a promising approach. In terms of effort to construct, the technique proved to be tractable. The

author was able to construct the robustness case in approximately 100 hours with input from relevant technical specialists.

The use of Goal Structuring Notation as a diagrammatic technique meant that the outputs were easily understood even by those not familiar with safety cases. The technique appeared to add value not just in providing structure to existing measures e.g. specifications, statements of work, but also highlighted areas where additional measures needed to be taken. A further benefit was that it provided a mechanism to ensure that evidence e.g. results of previous testing, is systematically gathered and reviewed. In complex systems, especially those with legacy components, it can be easily but wrongly assumed that previous testing has been successfully completed. It may be the case that a legacy part of a system may not have achieved its specified behaviour in a manner which becomes critical in the new system context resulting in system failures.

However it is important to evaluate the method with another application to prove it is more widely applicable, as for example the inherent modularity of MOST infotainment systems may favour the approach and make the breakdown into argument modules easier to achieve than on a more heterogeneous system. This will be investigated in the next chapter. As part of this work definitions of relevant parameters such as “acceptably robust” and “normal operation” will be developed.

5.5.2 Infotainment Robustness Modelling Initial Conclusions

An approach has been shown for robustness modelling of automotive electrical systems of systems based upon focusing on the ability to function rather than the functions themselves. This abstraction is necessary to make the development and use of the models of large scale systems tractable. While the customer functions themselves were not modeled the

timing of their availability to the customer could be derived from the model which is critical to customer satisfaction.

The robustness model gave useful results and an ability to explore design alternatives which would not be economically feasible with the physical system which would require large amounts of hardware and software changes across several ECUs for uncertain benefit. The work has shown it is possible to model the behaviour of MOST control channel to investigate the performance effect of key parameters with a view to their optimisation. In doing this it has highlighted the non-linear and emergent behaviour of the system deriving results which would not be possible to do through a manual analysis. The results gave a number of options for improvements which could be further explored and evaluated on a cost benefit basis before being physically manifested. It was also possible with the model to do grey box testing and observe states of the model such as utilization of buffers and initialisation states prior to communicating which was not possible with physical parts.

Application of the modelling principles to a real system has shown that a high degree of effort can be required to get models which embody critical features for robustness analysis. The infotainment robustness model took approximately 6 man months to develop, however it is envisaged that with less learning required subsequent models could be developed in a shorter timeframe, even on a different type of system. This will be confirmed in the evaluation study described in the next chapter.

5.6 Generic Methodology Description for New Methods

In this section the core generic content of the methods is abstracted from the Infotainment case study and described to enable the use of the methods in other applications.

5.6.1 Generic Methodology for Robustness Cases

Robustness cases are structured arguments for why a system will not exhibit service failures in the presence of errors caused by external faults. Robustness Cases make use of Goal Structuring Notation (GSN) and no changes are made in the use of the notation from that described in the GSN Community Standard (GSN Community 2011). The selection of tools is not prescriptive but they must support modular arguments. The robustness case should be initiated at the concept phase of the system design to identify design requirements and potentially existing design solutions which can be reused. The author of the robustness case should have knowledge of automotive electronics but does not have to be a domain expert providing access to domain expertise is available. A six step process for the development of robustness cases is illustrated in Figure 5.36 and described in the subsequent section.

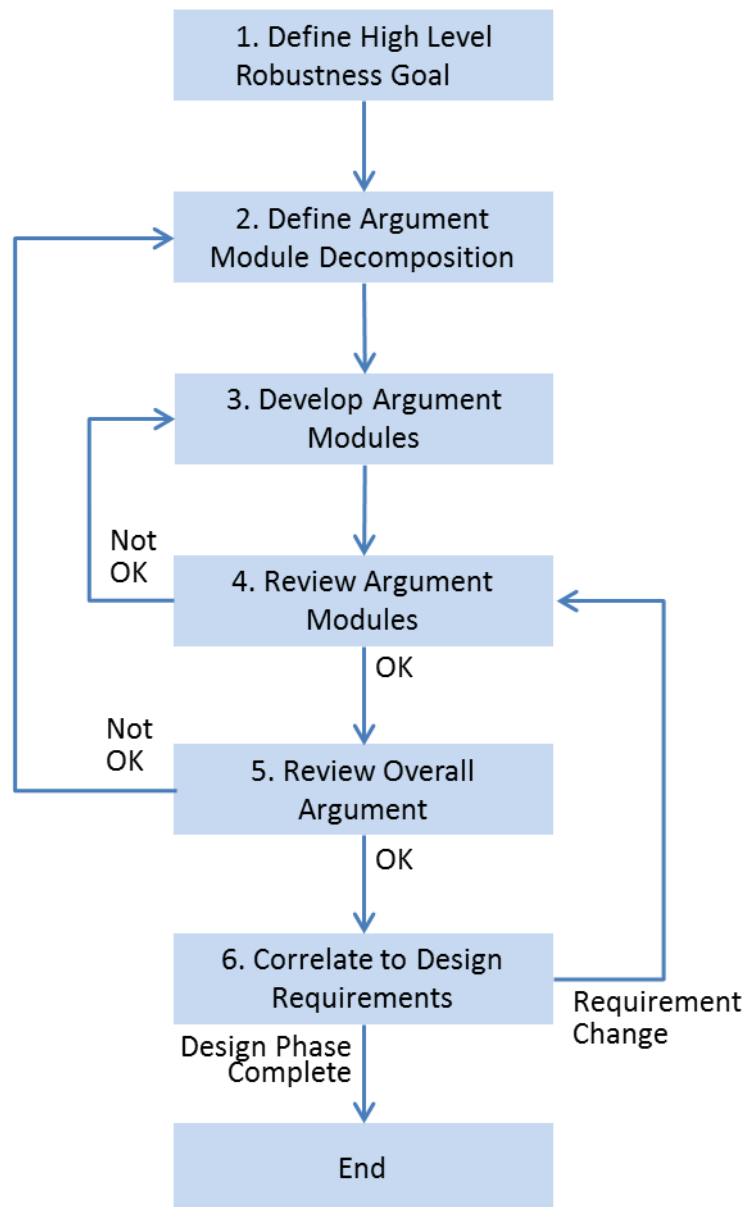


Figure 5.36 Process steps in developing Modular Robustness Case

Step 1. Define High Level Robustness Goal

The first step is to define the high level robustness goal, robustness being defined as freedom from service failures in the presence of errors due to external faults. Crucially these service failures should consider the wider set of dependability attributes not being restricted to safety (the system under consideration may have not safety critical functions) but including availability and reliability as the most significant. To do this requires an acceptable level of

service in the presence errors due to external failures to be defined. This may not be possible to define at an early stage and so an assumption may have to be made and subsequently followed up. As robustness is with respect to external failures it is necessary to be clear on what is the boundary of the system. The boundary of the system, or an individual part of a system of systems, can be considered to be the programmable device with responsibility for controlling delivery of a particular service and any dedicated sensors and actuators.

Step 2. Define Argument Module Decomposition

The next step is to use a top level argument to decompose the high level robustness goal into a number of argument modules. This should use the principles used in modular safety cases of: high cohesion and low coupling, supporting work division and contractual boundaries, supporting future expansion and isolating change. For automotive use a particular concern is responsibilities split between the OEM and the supplier. Argument modules should address specific areas known to be problematic on the basis of domain knowledge, e.g. analysis of previous failures such as the case studies in Chapter 3.

Step 3. Develop Argument Modules

Each of the argument modules identified in the top level argument should be developed by breaking down each argument specific top-level goal into strategies, sub-goals and solutions which would form the evidence for achieving each sub-goal. These require domain knowledge to create and so domain experts in the specific area covered by the argument module should be engaged. In most cases existing solutions will exist so part of the argument may be developed through a bottom up approach inferring sub-goals from the solutions. As the application of the method matures then there will be existing argument modules which can be referred to and re-used if they have been proven to be successful.

Step 4. Review Argument Modules

When an argument module has been created it should be reviewed through a suitable peer-review process in a similar manner to other design artefacts. If weaknesses in the argument are found then the argument should be further developed.

Step 5. Review Overall Argument

The overall argument should also be peer reviewed to ensure there are no gaps in the top level argument. Once again if weaknesses are found then the argument should be further developed either through adding new argument modules or augmenting existing ones.

Step 6. Correlate to Design Requirements

Once the argument structure has passed through review then it is necessary to ensure that the requirements within the arguments are reflected in the design requirements. Typically design requirements and related verification methods are held in a database within a requirements management tool such as DOORS. This can be used to ensure design requirements are fulfilled and solution evidence is gathered as part of the normal design and development process. A field can be added to show which requirements are related to the robustness case so that monitoring can be done in case of requirements changes which may impact the robustness case. Due to this need to monitor the impact of changes the robustness case remains a live document until the end of the design phase when requirements are frozen.

5.6.2 Generic Methodology for Robustness Modelling

In this section the approach to robustness modelling developed within this chapter is explained as a generic methodology with wider applicability. The approach focuses on modelling the availability of resources required for service delivery, in particular processing and communication, rather than modelling the specific functionality of the service. This model should be ideally created as soon as the initial deployment architecture for the service under consideration is available. This deployment architecture should define the Electronic Control Units (ECUs) involved in the service delivery, the interaction between them and the communication networks. For robustness considerations it is essential to understand which ECUs are providers of inputs to the ECU controlling the service delivery.

Figure 5.37 illustrates the generic structure of the robustness model. The ECU controlling the service delivery and ECUs which are providers of inputs are modelled as interlinked state machines. The ECUs have inputs associated with timing, user inputs, system operating mode and power supply which the case studies showed were all critical parameters in robustness related issues. There is also a block within the model concerned with distributed functions based on availability of processing and communication resources. Each block is described in more detail in the subsequent paragraphs.

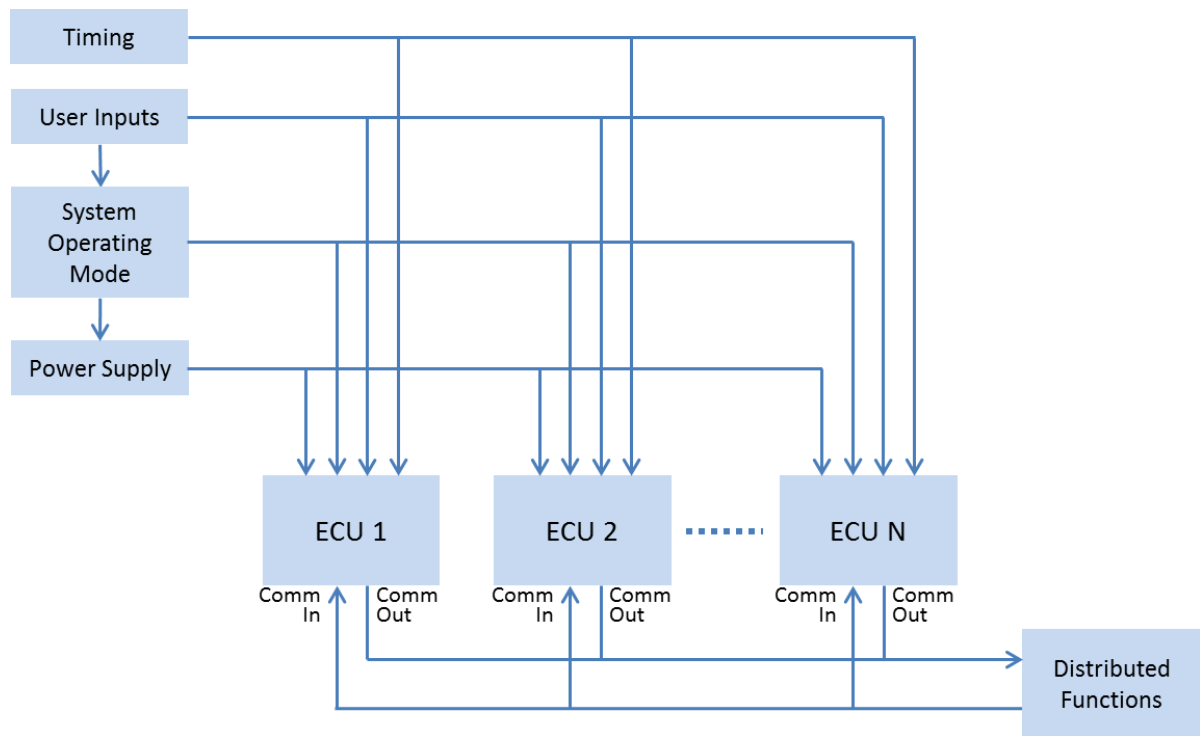


Figure 5.37 Generic structure of Robustness Model

The timing block provides a common clock signal to the ECU blocks and should be set to a higher level of resolution than expected response times within the system. For the initial development this was set to 1 ms and for the infotainment study reduced to 1 μ s. This is then used by each individual ECU block as a shared definition of time and to ensure their state machines can respond sufficiently quickly noting there will be need to model the response of events within a statemachine to occur more slowly to reflect actual system responses.

User inputs which affect both the specific service under consideration and also the wider system operating mode should be modelled. Typically these are modelled as switches between binary values or constant blocks giving parametric values. For a vehicle system the operating mode (powermode) is typically influenced by wake up events such as door unlocks, door opening and pressing of start button.

Using these inputs the System Operating Mode determines the current operating mode of the vehicle (Powermode) and then sends this to the ECUs which will each take appropriate initialisation or shut-down responses.

The power supply voltage level was found to be a critical parameter in the case studies particularly with respect to short low voltage fluctuations during starting of the engine. The power supply block provides a parameter-based, continuous analogue signal of the voltage level to the ECUs and can simulate voltage fluctuations when the powermode indicates an engine starting state.

Figure 5.38 illustrates the inputs and outputs to a generic ECU Block. The timing input is used to trigger the state machine and to determine the duration of events. The supply voltage input is used to determine whether the ECU is powered or unpowered depending on its minimum operating voltage parameter. The System Operating Mode input is used to determine whether the ECU should power up or shut down. Significant parameters of the ECU which can impact availability and are subject to variation should be set by constant blocks which user can configure. Typically this should include: minimum operating voltage, initialisation time, shutdown time, cycle times for performing specific tasks e.g. servicing communication interfaces and any built in delays within the system to wait for external events to occur. The external reset causes the ECU to re-initialise to be able to fault inject to simulate the effect of internal ECU faults such as watchdog resets. The external wake-up causes the ECU to initialise to facilitate fault injection of spurious ECU wake-ups. Comms In is used for other input signals needed for the ECU platform to initialise and shut-down. The Comms Out is the data which the ECU provides to other ECUs required for the delivery of the required service. Where possible the actual messages should be abstracted into generic classes represented by an enumerated value of the message types. The node status is used to communicate and externally observe the internal state of the ECU, in particular its

availability in terms of ability to function and communicate. In a model it is also possible to make observable parameters which in an actual system would not be, such as ECU initialisation states or buffer utilisation in the case of the MOST infotainment model.

The Comms Out and the Node Status information are used by the distributed system block to determine the availability of the particular service under consideration. The block can also be used to implement distributed functions for arbitrating communications to determine which of the messages that the ECUs attempt to send will be successful.

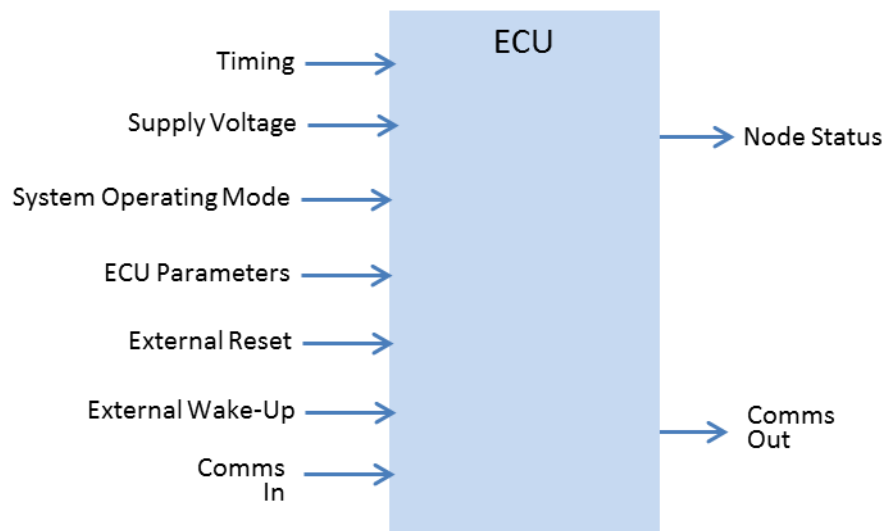


Figure 5.38 Generic ECU block inputs and outputs

Figure 5.39 shows the internal structure of a generic ECU block illustrating the generic states and transitions between them. If the voltage is below the minimum operating voltage the ECU will be in the Unpowered state. When voltage is above minimum operating voltage then the ECU will go into either asleep or initialisation state depending on inputs such as system operating mode. In the sleep state the ECU uses minimum power until it receives a wake-up trigger to initialise. The initialisation state represents the time delay for boot-up of microprocessors and communication devices which is an externally configurable parameter.

After this is complete an ECU is considered to be in its running state and able to execute communication functions and application layer interactions which may be necessary before the service can be provided. Some ECUs have shutdown routines, e.g. data-storage, that they must perform before going to sleep mode. Power interruption during shut-down can cause issues e.g. failure to write-back to non-dynamic memory, which should be captured in the model by a different Node Status value.

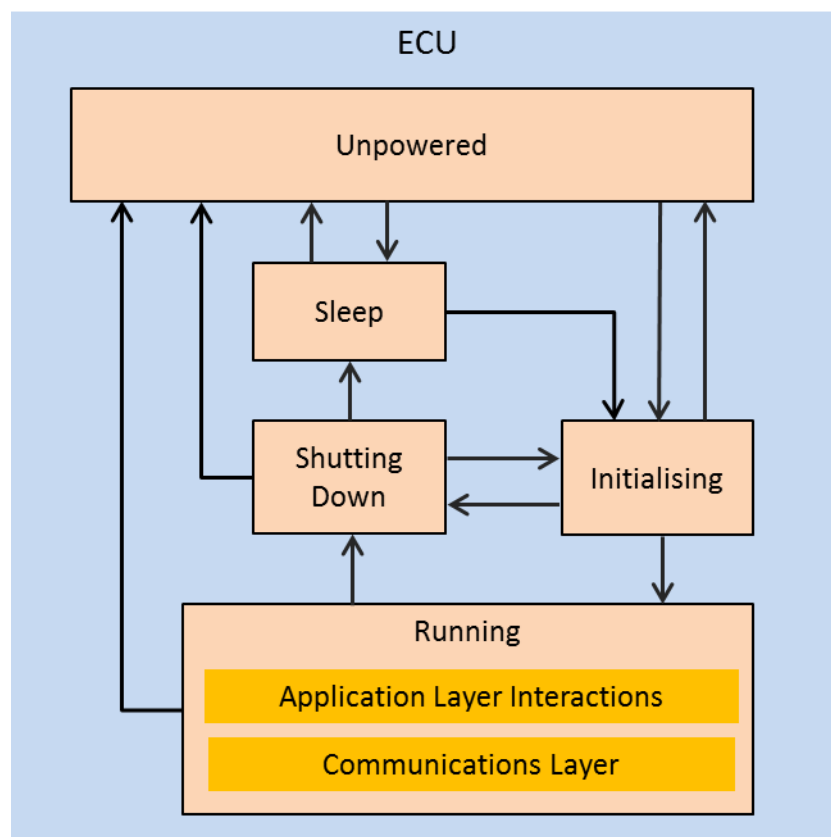


Figure 5.39 Generic ECU block internal structure

The level of detail of the modelling of the application layer interface interactions and communications layer will vary depending on the characteristics of the specific system as was seen in the generic model and the MOST initialisation models.

The testing of the model should be determined by the requirements coming from the Robustness Case. This will be explored further through application of the methods in combination in the next chapter.

Chapter 6 - Development of New Robustness Techniques through Application to a Hybrid Propulsion Control System

6.1 Introduction

In the previous chapter the new proposed techniques of robustness cases and robustness modelling were developed through application to an infotainment system. In this chapter the techniques will be evaluated and confirmed to be more widely applicable through applying them to the system initialisation of a hybrid propulsion control system. Firstly the hybrid system initialisation process is discussed in terms of the key objectives and its technical implementation, illustrating the level of complexity underlying a simple high level requirement. Then the development of the robustness case is described for the hybrid system initialisation including the identification of robustness parameters, the identification of robustness requirements and requirements for model-based testing. Following this the development of the robustness model for hybrid system initialisation is described including the testing process and results. Conclusions are given on the differences from the previous application and hence the transferability of the methods and also on utility of the individual techniques. Finally extensions to the generic methods arising from the evaluation are described including how the techniques can be used in a complementary manner as part of the overall design for robustness framework.

6.2 Hybrid Propulsion System Initialisation

6.2.1 Hybrid propulsion system and robustness parameter analysis

The case study selected for evaluating the proposed new methods for robust design was that of a parallel hybrid electric propulsion system. This was selected on the basis of being an

example of complex system with many interacting parts that the author had access to the underlying design information but is different to the infotainment application used to develop the methods. A significant difference is that unlike the infotainment system the constituent parts are non-homogeneous, this is important in showing the methods are more generically applicable. While the hybrid system has safety-related requirements the case study focused on robustness requirements relating to system availability.

A simplified driveline architecture for a parallel hybrid electric vehicle is shown in Figure 6.1. In the case study example an electric motor is incorporated within the transmission between two clutches such that the drive propelling the vehicle can be delivered by the internal combustion engine (ICE) or the electric motor (EM) or a combination of both in parallel. The electric motor is powered from a high-voltage battery via a power inverter. The battery is charged from the electric motor as a result of either regenerative braking or by using energy from the internal combustion engine.

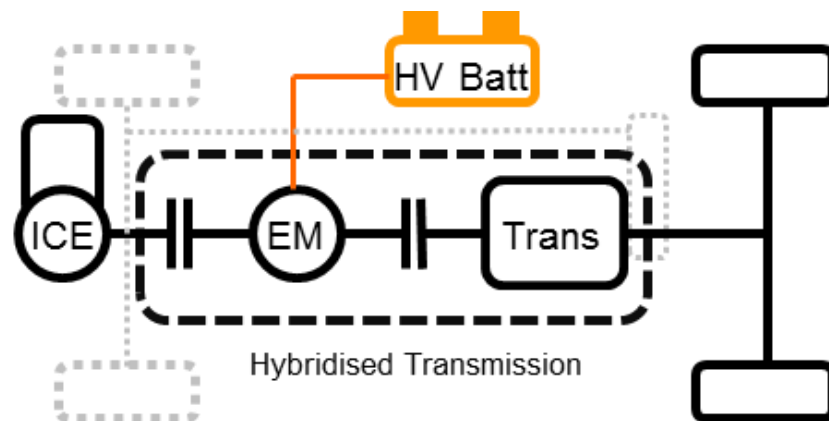


Figure 6.1 Parallel Hybrid Electric Vehicle driveline architecture

An analysis of robustness factors impacting the HEV system was conducted using a Parameter Diagram (P-Diagram) approach (Fritzsche 2006). The P diagram considers the

inputs, ideal outputs, error states, control factors and noise factors in a system. The P diagram that was developed for the hybrid system is shown in Figure 6.2.

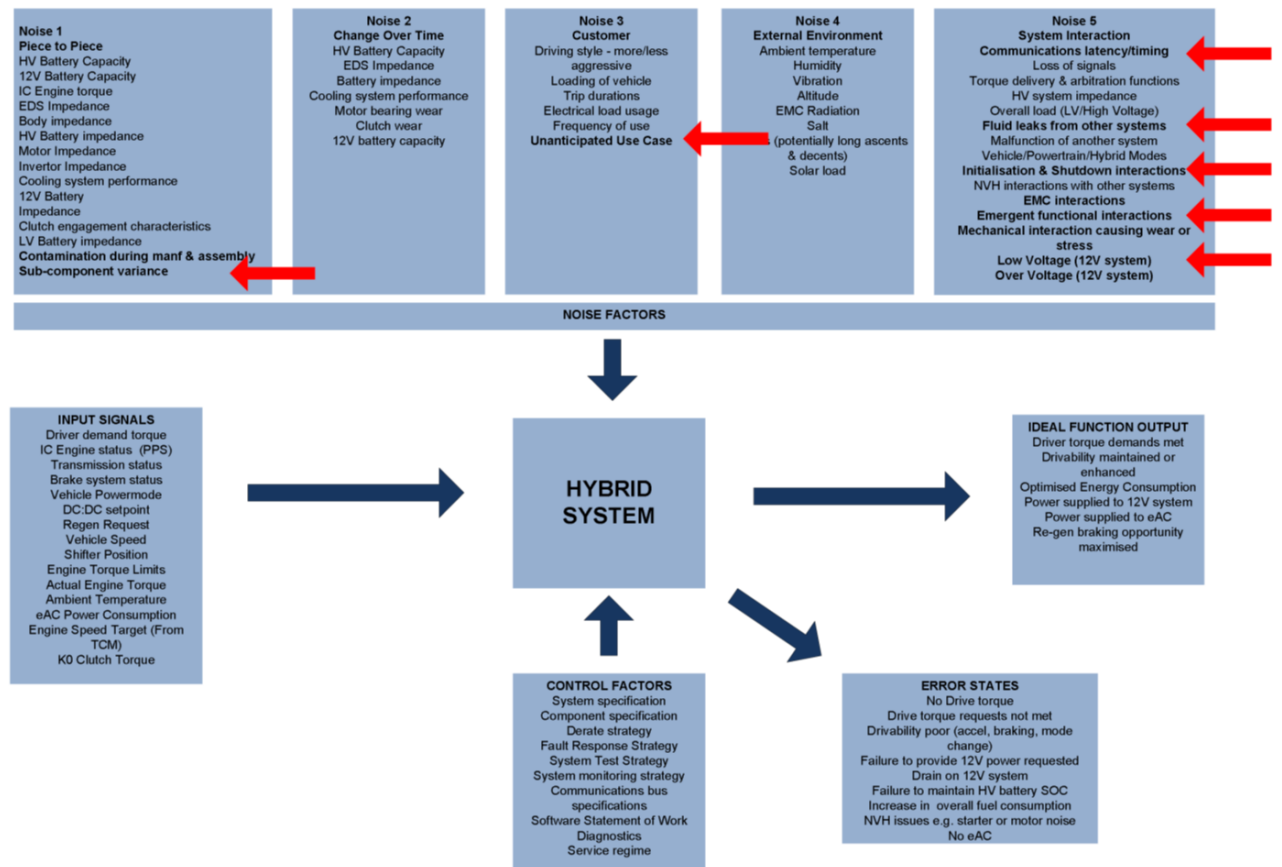


Figure 6.2 Parameter Diagram (P-Diagram) for hybrid system

Of particular interest in the context of robustness are the noise factors. The P-diagram classifies the noise factors into; piece to piece variation, changes over time, customer factors, external environment and system interaction. From Figure 6.3 it can be seen that for a complex system there are considerable variety of noise factors. Many of these are possible to address at a component level via traditional quality techniques, those that are of most interest for this study are those noise factors relating to system interaction. On the basis of the findings of the case study review in Chapter 3 the noise factors selected for further investigation using the new methods was that of initialisation and shut down interactions,

sub-component variation, low voltage, delayed communication, user interaction and communications timing.

6.2.2 Hybrid propulsion System Initialisation

From a user's perspective the hybrid system initialisation is simply the time between pressing the start button and the vehicle being ready to put into gear and drive off. In a hybrid electric vehicle it may initialize into one of 2 modes either with the internal combustion engine on (referred to as a "Combustion Start") or by activating the high voltage system to allow the vehicle to drive off using the electric motor (referred to as "HEV Standby") with the internal combustion engine off until required. In practice this is influenced by the state of the vehicle powermode (the virtual key position), the status of the engine and the status of the high voltage propulsion system which in turn are each determined by a large number of parameters ranging from security authorization checks to high voltage system integrity checks. It is also necessary to consider the initialisation of the Electronic Control units (ECUs) which provide the processing platforms for determining and communicating the states of the relevant system parts.

Figure 6.3 shows the principal components involved in the initialisation of the hybrid propulsion system under investigation.

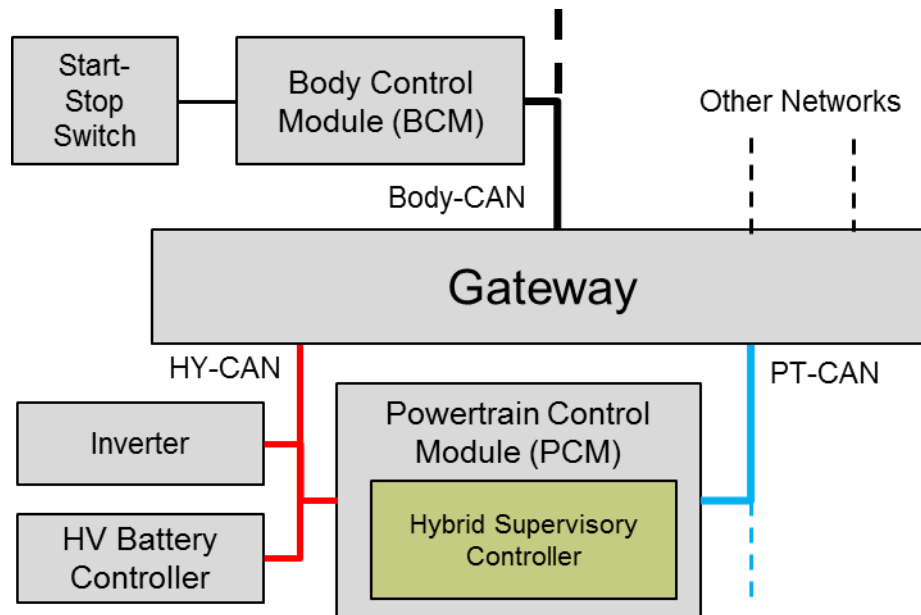


Figure 6.3 Simplified network architecture diagram of HEV System

The body control module receives the signal from the start switch and based on this and other factors determines the overall power management state of the vehicle. The Powertrain Control Module (PCM) determines the powertrain status and hosts the hybrid supervisory control which monitors the status from other hybrid controllers during initialisation and determines the mode into which the hybrid system will initialise. The high voltage battery management system controls the high voltage battery system including functions such as checking system isolation, checking high voltage system integrity via an interlock loop, checking contactor status and ultimately closing contactors to connect the high voltage power source. The hybrid power inverter controls the motor and the DC to DC converter that provides energy to the 12 volt system and also performs checks on the integrity of the high voltage system.

The communications between the Body Control Module (BCM) and Powertrain Control Module (PCM) go through the gateway module, then are via a high speed CAN bus. Additionally the BCM controls a hardwired ignition signal which is received by the PCM. A

second high speed CAN bus connects the PCM with the hybrid components. Other communication buses not relevant to hybrid system initialisation have been omitted in Figure 6.3 for clarity.

The individual parts of the system demonstrate its non-homogeneous nature being defined across a number of different organizations in a number of different types of design specifications and delivered through different routes. The power mode management in the BCM is defined in a state chart specified by the Electrical Department and is largely the same as for non-hybrid vehicles. The power management function is auto-coded by the BCM supplier from the in-house developed model. The power pack status is defined in a paper specification which includes a state chart drawing developed by the Powertrain Department with an implementation by the PCM supplier. It is largely the same as for non-hybrid vehicles with stop-start functionality. The hybrid system control is defined through SysML drawings (block diagrams, activity diagrams, and state machines) by the hybrid systems team but not in an executable fashion. The hybrid supervisory control in the PCM is delivered by an in-house team through model based auto-coding. The functions in the inverter and battery controller are fully supplier delivered based on textual specifications.

6.3 Hybrid Propulsion System Initialisation Robustness Case

Development

This section describes the development of robustness case for the initialisation of the hybrid propulsion system. The development of the robustness case took approximately 50 hours focusing on most relevant areas e.g. at a system rather than component level. The full robustness case is shown in Appendix B with key parts highlighted in this section to explain its construction.

6.3.1 Top Level Goal and Decomposition Strategy

Goal Structuring Notation (GSN) arguments decompose a top level goal in a particular context through strategies and sub-goals to solutions with appropriate justifications. In the case of the Hybrid initialisation (shown in Figure 6.4) the top level goal is “*the Hybrid system initialises & shuts down robustly*”.

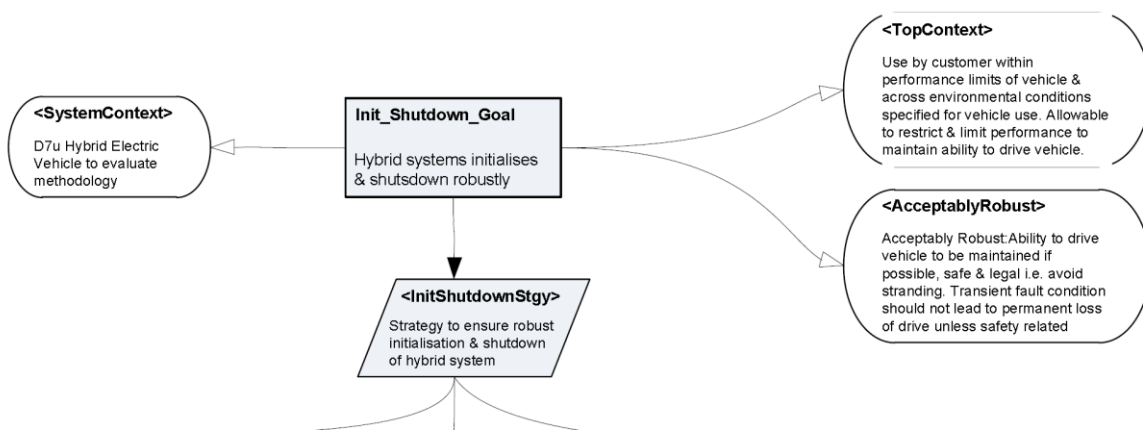


Figure 6.4 Top level goal of GSN robustness case for hybrid system initialisation

This goal needs to be put in context of the system, the vehicle use and a definition of “*acceptably robust*”.. The vehicle use context is defined as “Use by customer within performance limits of vehicle & across environmental conditions specified for vehicle use”. This definition seeks to avoid over engineering while not placing restrictions on how the customer uses the vehicle within these limits.

Acceptably robust is defined as “*Ability to drive vehicle to be maintained if possible, safe & legal i.e. avoid stranding, transient fault condition should not lead to permanent loss of drive unless safety related*”. This encompasses the concepts of resilience through

maximising availability and ensuring the recoverability from error states that have previously been discussed in Chapter 2 as vitally important for the robustness of complex systems.

The strategy for meeting the top level goal decomposes the top level goal into sub-goals at a system design, component design and validation levels as shown in Figure 6.5. These form the top level goals for a series of argument modules. The strategy for decomposition at this level was based on organizational responsibilities and so split into sub-goals for systems design, component design and verification. Colour coding with a legend was introduced to identify ownership.

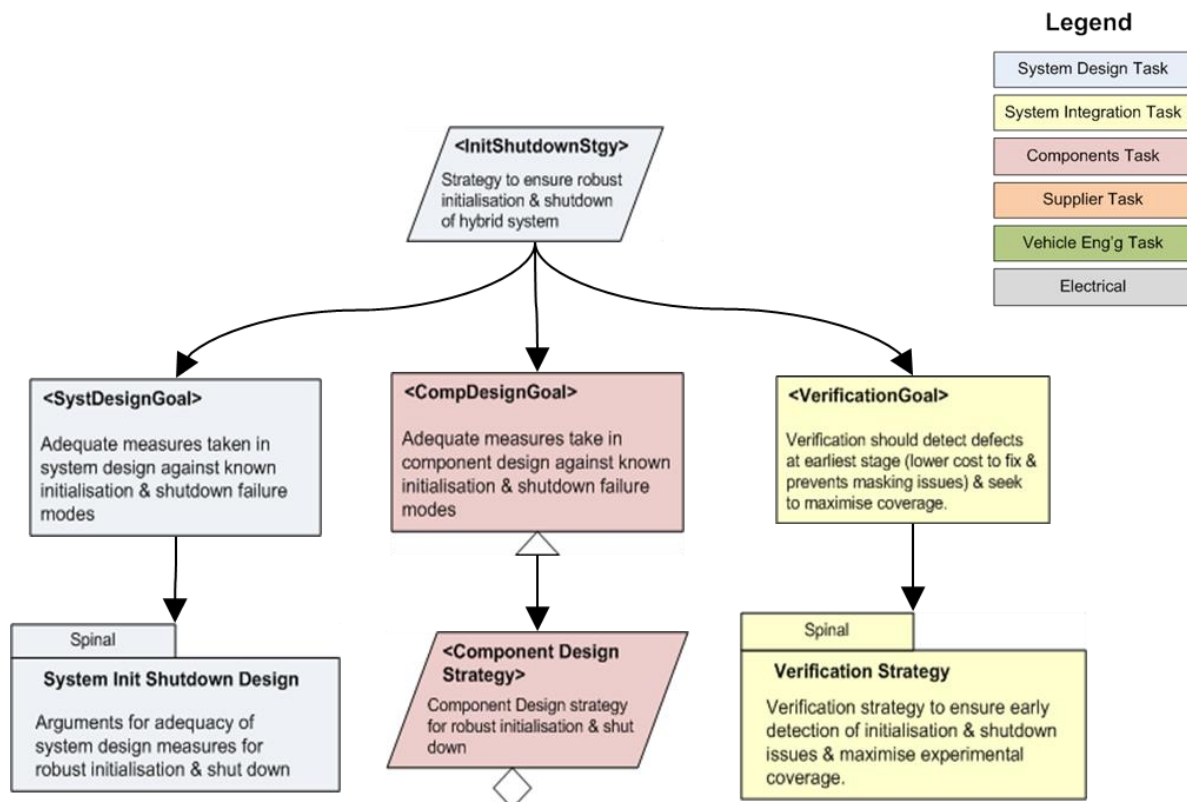


Figure 6.5. Decomposition strategy for top level goal of GSN robustness case for hybrid system initialisation

The component design strategy was not further developed at this stage but the focus was put on developing the system design and the verification strategy as these would feed into the robustness model and its testing. These argument modules will be described in the subsequent sections.

6.3.2 System Design Robustness Argument

The hybrid system initialisation design goal was decomposed to 10 sub-goals each of which formed argument modules for further decomposition. The strategy for this was based on the system design specifying adequate measures against known initialisation and shut-down failures. This is illustrated in Figure 6.6 to show the structure of the decomposition, the details are subsequently described.

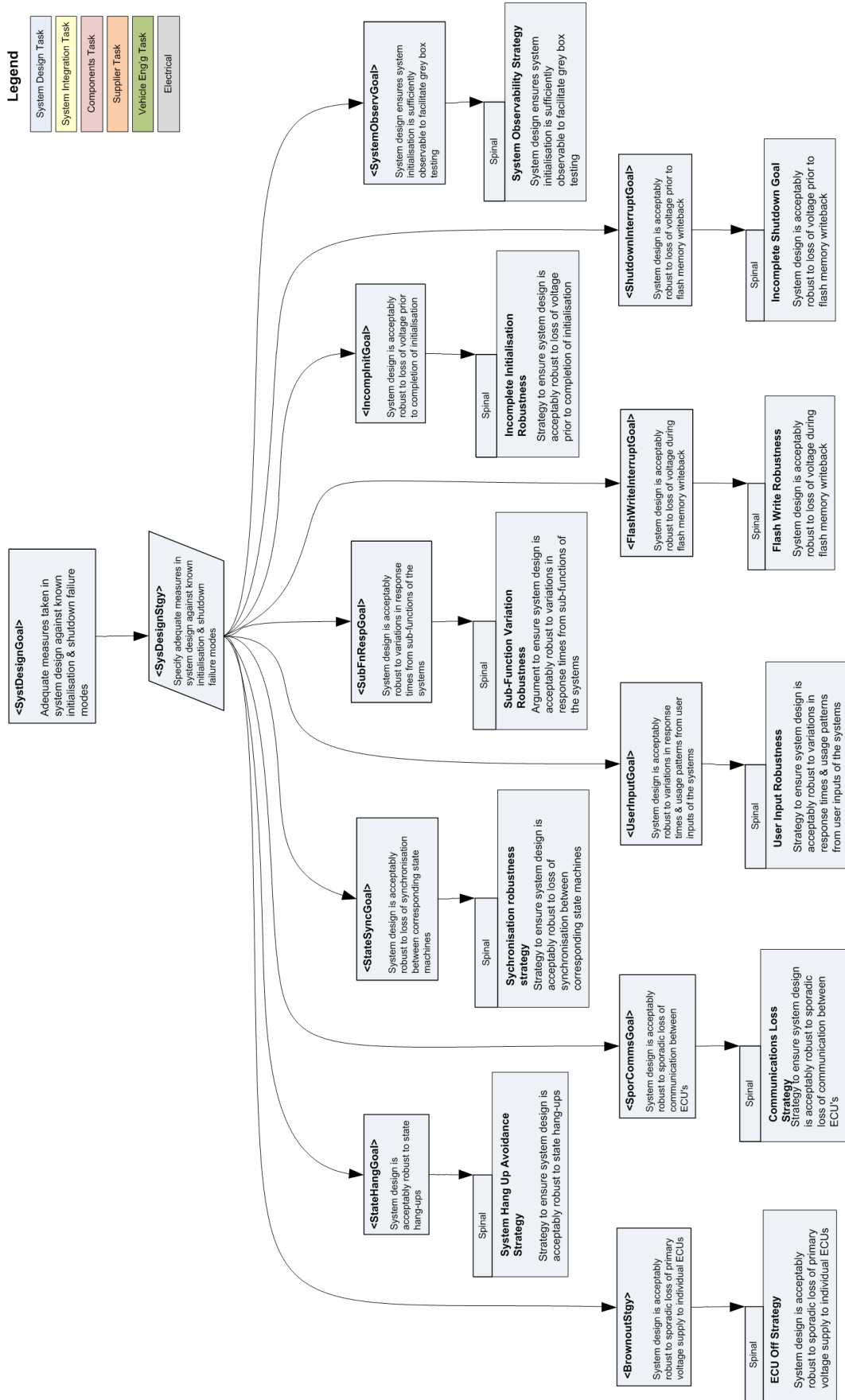


Figure 6.6 Decomposition of system design goal

The 10 sub-goals were mainly defined based on knowledge of generic causes of failure identified in Chapter 2 cover issues such as brownouts, state hang-up, loss of communications, loss of synchronization between parts of the system, variations in user inputs and usage patterns, variations in response time from system parts, flash memory write robustness and incomplete initialisation or shutdown. A sub-goal added to enable correlating the robustness model and actual system testing was that the design should ensure that system initialisation is sufficiently observable to facilitate grey box testing. The intent of this is that even if a supplier is implementing the design for which the detail is not known, i.e. as a black box, then state variables should be made available as parameters to enable observability during testing.

Each of the 10 sub-goals were developed as argument modules to a solution level. An example of one of these arguments for the state hang-up goal is shown in Figure 6.7.

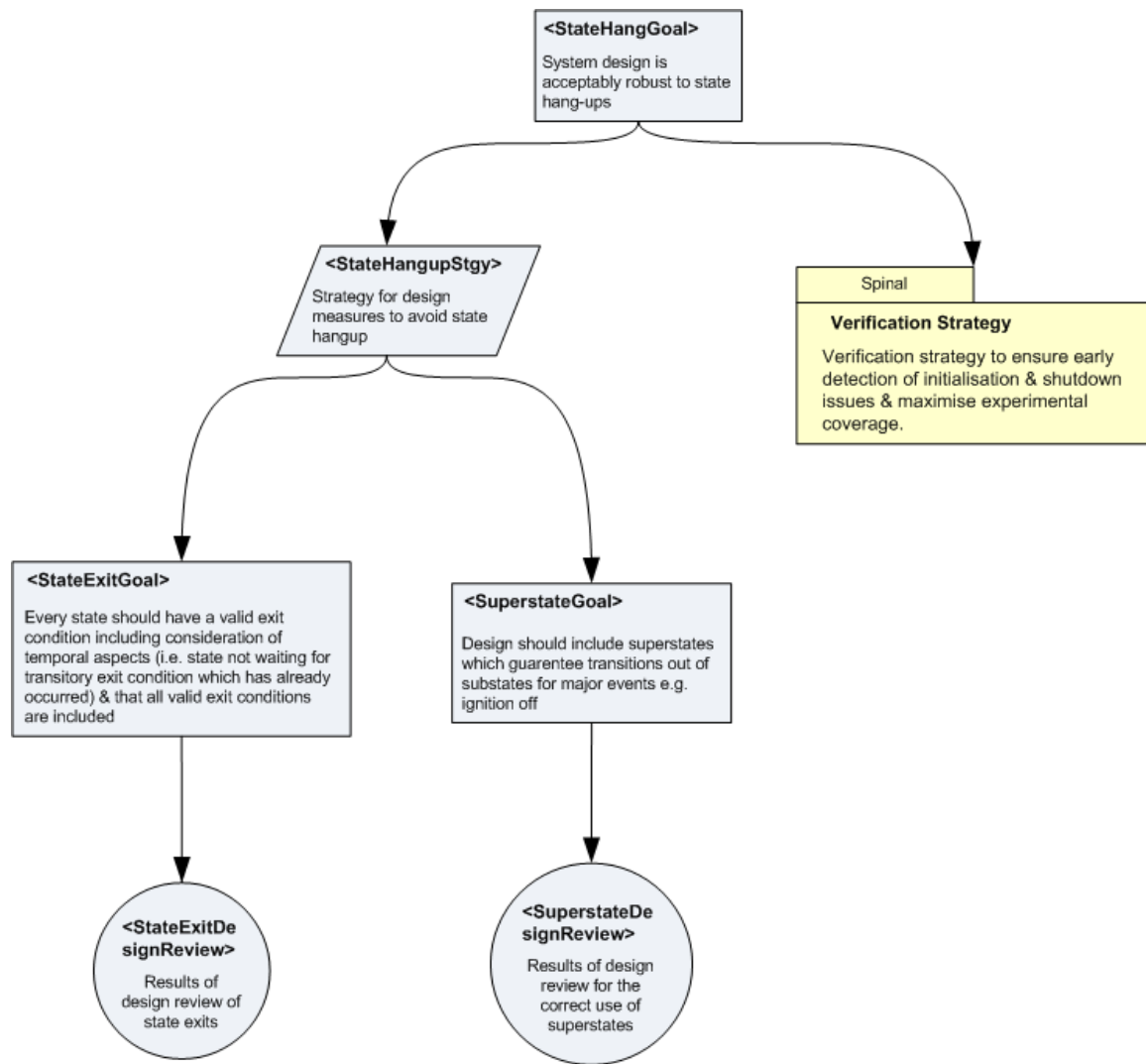


Figure 6.7 Robustness argument module for state hang-up system design goal

As can be seen in Figure 6.7 the argument breaks down more explicit design goals which are to be evidenced by design review and through the verification strategy. The breakdown of the argument required subject matter expert inputs either through direct consultation or from design guidelines.

As has been observed in safety case development (Palin and Habli, 2010) it was found that there are often distinct classes of robustness goals whereby patterns can be used to codify and reuse design solutions to these objectives, an example of this being the argument for

robustness to incomplete shutdown shown in Appendix 2 which has the potential to be re-used on a wider basis.

6.3.3 System Verification Robustness Argument

In the System Verification Robustness Argument detailed verification requirements are derived to provide the evidence of meeting the robustness sub-goal of “*Verification should detect defects at earliest stage (lower cost to fix and prevents masking issues) and seek to maximise coverage*”. The strategy to achieve this robustness verification goal was through multi-level testing using models, components system and vehicle level tests, the argument module for this is shown in Figure 6.8.

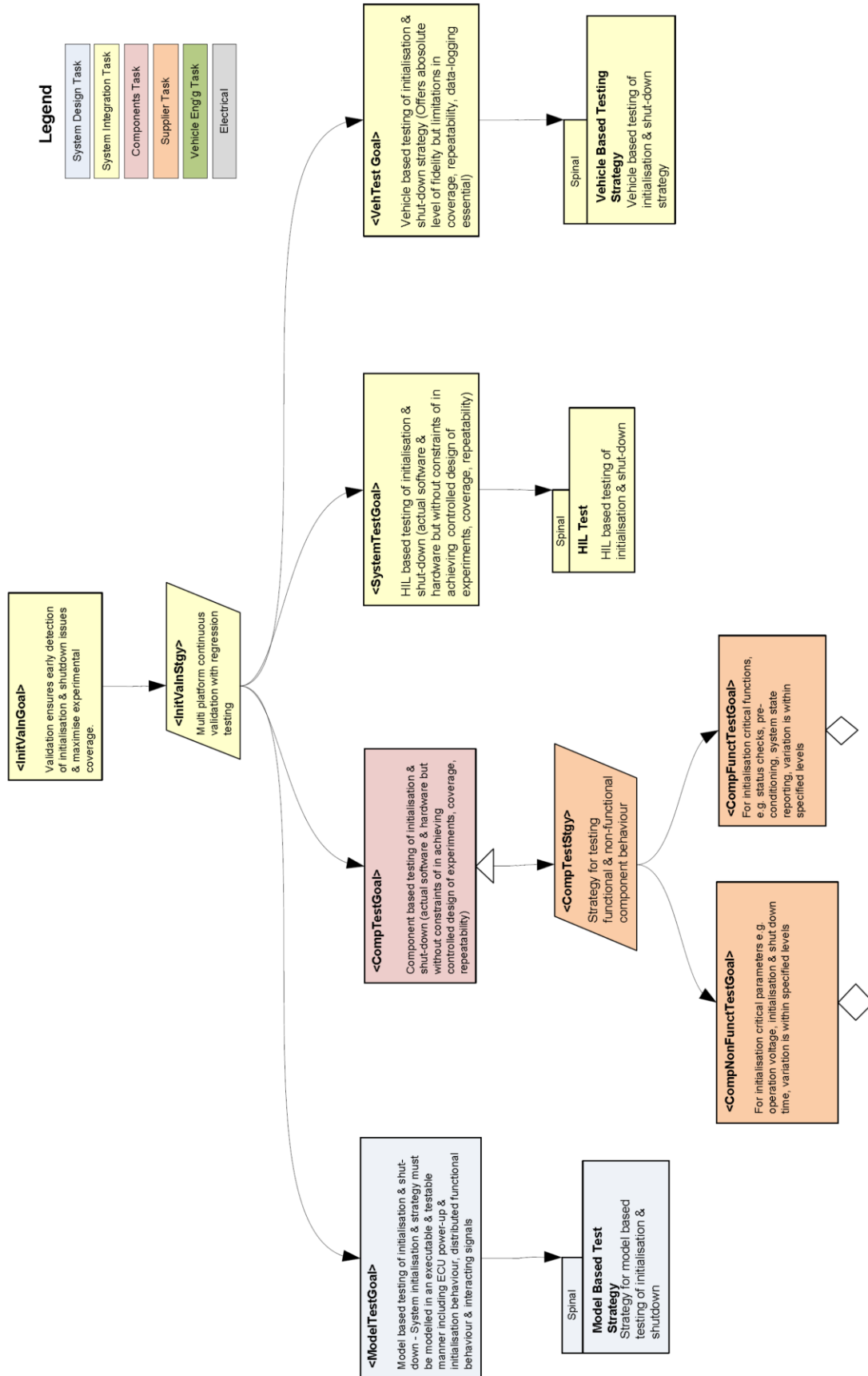


Figure 6.8 System initialisation verification robustness argument

The first level of decomposition sets sub-goals for model, component, hardware in loop and vehicle based testing each of which has argument modules. The argument module defining the model testing strategy will be discussed later in this chapter as this was the basis for the testing of the robustness model.

6.4 Hybrid Propulsion System Initialisation Robustness Model

Development and Testing

This section describes the development and testing the hybrid propulsion system initialisation model.

6.4.1 Hybrid Propulsion System Initialisation Robustness Model Development

The hybrid propulsion system robustness model was based on the generic approach that has been developed for the robustness modelling of non-functional properties, e.g. operating voltage, initialization timing, and operating modes, relating to the ability of the system to function as opposed to the function itself described in the previous chapter. This approach models a distributed system as a series of interlinked state machines whose state is dependent not only upon signals received from other nodes but also, more fundamentally, factors that affect the ability of the node to be able to receive and respond to control signals. The hybrid propulsion system initialisation model was developed in Matlab Simulink and Stateflow and is illustrated in Figure 6.9. The model took approximately 250 man hours to develop and test.

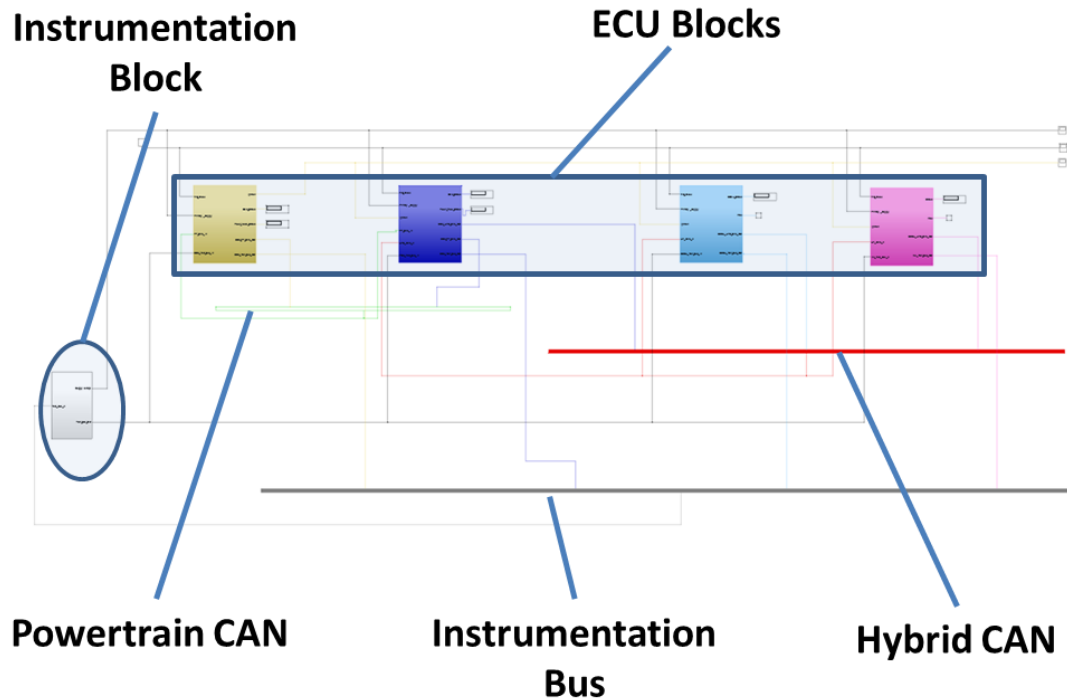


Figure 6.9. Hybrid propulsion system initialisation model - top level

The model at the top level consists of the individual ECUs described earlier with the exception of the gateway module which has been abstracted as this has no functionality other than to relay messages, the communication buses and the power-supply to the ECUs. An instrumentation block and signal bus has been included to automate running of test scenarios and aggregate data collection. Within each ECU block there is a master state machine for the operational mode of the ECU with states for unpowered, initializing, running and shutting down (see Figure 6.10).

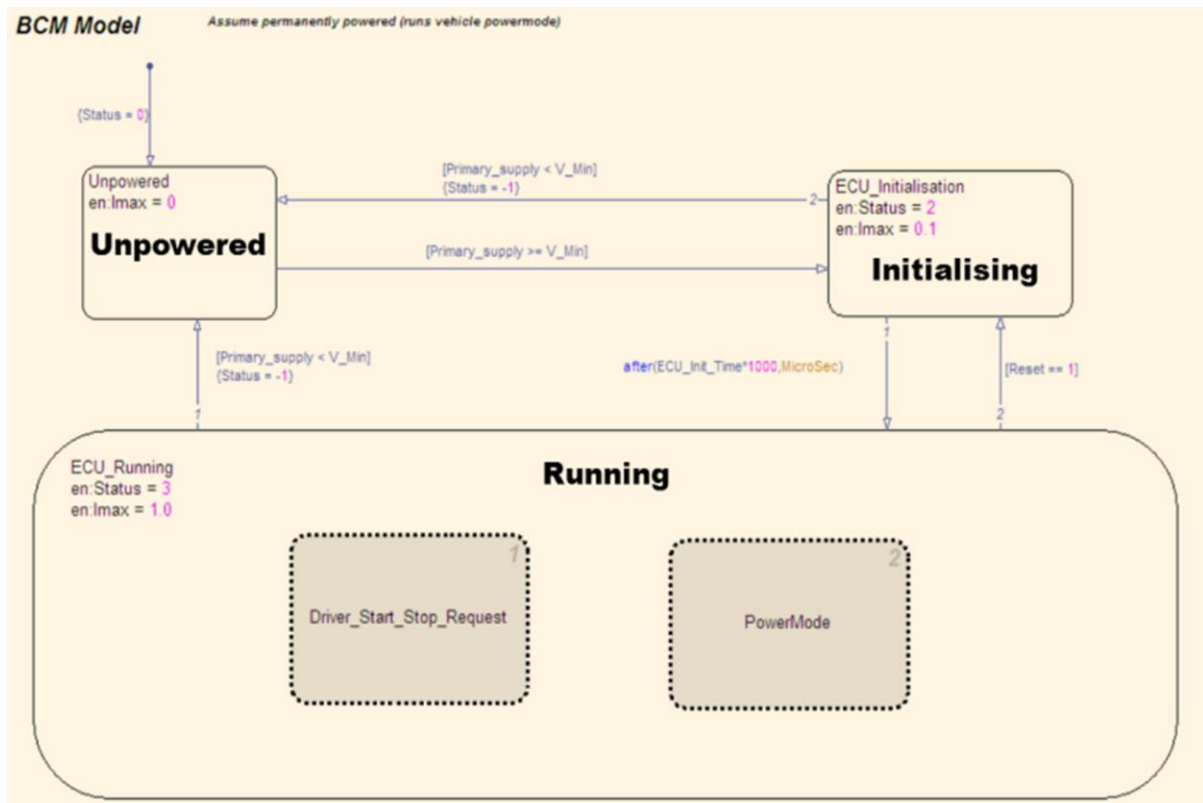


Figure 6.10 Hybrid propulsion system initialisation model - ECU state machine

In the running state there are underlying state machines for the behaviour of each the ECUs related to the initialisation strategy, for example driver start stop request and Powermode in the case of the BCM shown in Figure 6.10. This behaviour was modelled based on the individual specifications for each component.

It was also necessary to model some aspects of the plant systems, e.g. the engine. This is done by abstracting the plant system to state based behaviour and running it as a parallel state-machine to the control behaviour. This approach minimizes the additional complexity of the model by having simple behavioural model which will can be directly linked to the control behaviour. A more complex continuous behaviour would require a new Simulink block outside the ECU block. Part of the engine plant model is shown in Figure 6.11 showing

this abstraction to the key states of behaviour relevant to the system initialisation and their interaction. The high level states were whether the engine was stationary or moving, with sub-states for modes for stopped or stalled for stationary and stalling, cranking, running, and stopping for moving. It is important that abnormal behaviour is modeled, e.g. stalled in the case of the engine, as the resultant system behaviour could be different. It was also found through testing to ensure the plant model was able to fully exercise the controller, that to model some of the continuous behaviour of the engine it was necessary to break down the stopping state into two separate stages, one in which the engine could be recovered to running by refueling and one in which it had to stop and be started by the starter motor (cranking).

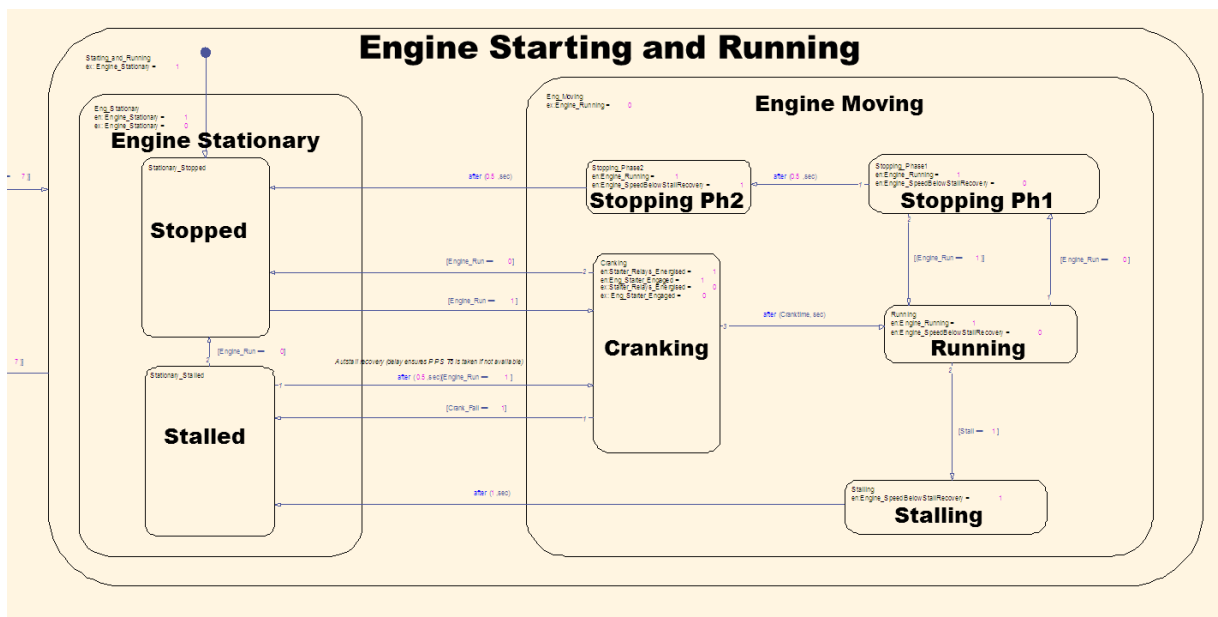


Figure 6.11 Hybrid propulsion system initialisation model – part of engine plant model state machine

It is important to model and make available control parameters to induce fault conditions identified in the robustness case, e.g. processor resets, under-voltage and other

fault responses. It is also necessary to make parameters available to allow control of factors which will be subject to variation. In the hybrid initialisation model this included time to complete particular functions which had physical interactions, e.g. checking integrity of high voltage system, and system timers which could be set through calibration.

To facilitate testing an “instrumented model” was developed with an additional test bus which was used to feed parameters from test scripts into the model and log behavioural parameters for analysis.

6.4.2 Hybrid Propulsion System Initialisation Robustness Model Testing

The robustness case (shown in Figure 6.12) identified four key areas of model based testing which were; reachability testing, parameter variation testing, fault injection testing and correlation testing. These were proposed based on the good practice for model development (reachability and correlation testing) and the activation patterns of robustness failures from case study experience (parameter variation and fault injection). These tests types are generally applicable and can be used as part of the generic method.

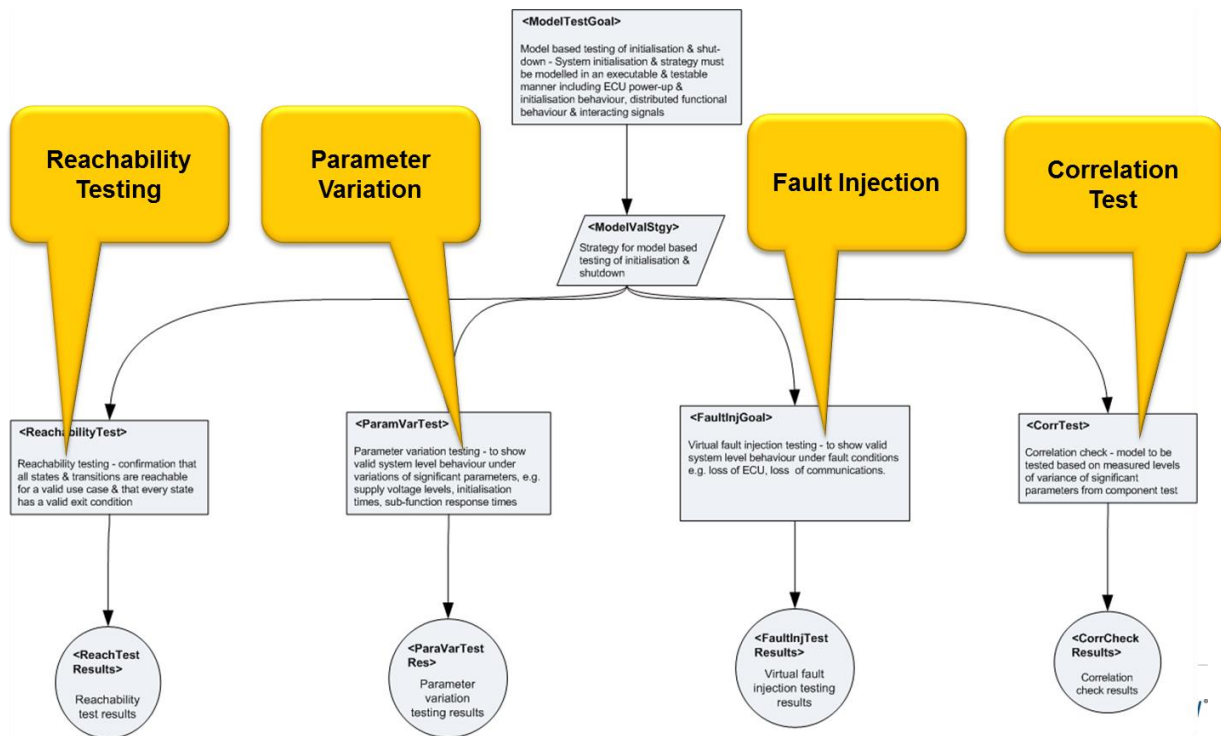


Figure 6.12 GSN robustness argument for model based testing

Reachability Testing

Reachability testing is analogous to MCDC (Modified Condition/Decision Coverage) testing of software which is intended to prove all entry, exit, statement and decision branches of the code can be invoked and that every condition of a decision can have an effect (Hayhurst et al.,2001). In a similar manner systematic testing was conducted to ensure that every state and transition within the model could be reached.

An example of test results is shown in Figure 6.13. Each state and transition of the statemachine under test was individually numbered (this can be generated automatically using documentation functions of Simulink) and the model inputs were manually adjusted to confirm each could be reached. This testing initially realized results relating to the completeness of the model, e.g. the plant model had to be enhanced to be able to fully exercise the model. Subsequently the testing did identify a design error (which was

concurrently detected by peer review), areas where it was necessary to precisely specify aspects of the implementation and a transition which had become redundant due to behaviour of another state machine. A specific example of an issue found during reachability testing was a state which triggered behaviour in another statemachine that could be reached with its exit conditions in place. This meant that it could be highly transitory and potentially cause a race condition between the state exit condition and the state machine with dependent behaviour detecting it had entered this state.

Figure 6.13 Reachability test results

The next set of tests related to parameter variation. Sweeps were conducted of a number of critical parameters to determine their impact on the system initialisation

performance, primarily the time to reach a ready to drive state and the mode into which the system initialised. Input parameter matrices were developed in Microsoft Excel and read into the instrumented model from mat files (data file format used by Matlab Simulink). The results were logged and transferred to MS Excel for analysis and presentation.

Figure 6.14 shows an example test result for sensitivity of the start time to the engine pre-start time where the engine is made ready for cranking by priming fuel and heating glow-plugs if required. The parameter sweep was performed in two conditions, one where the Hybrid system would start in HEV standby mode and the other where an engine start would be required by environmental conditions. In the first case the engine pre-start time makes no difference to the start time, as the engine is not started, but in the second after a minimum period it adds a linear delay.

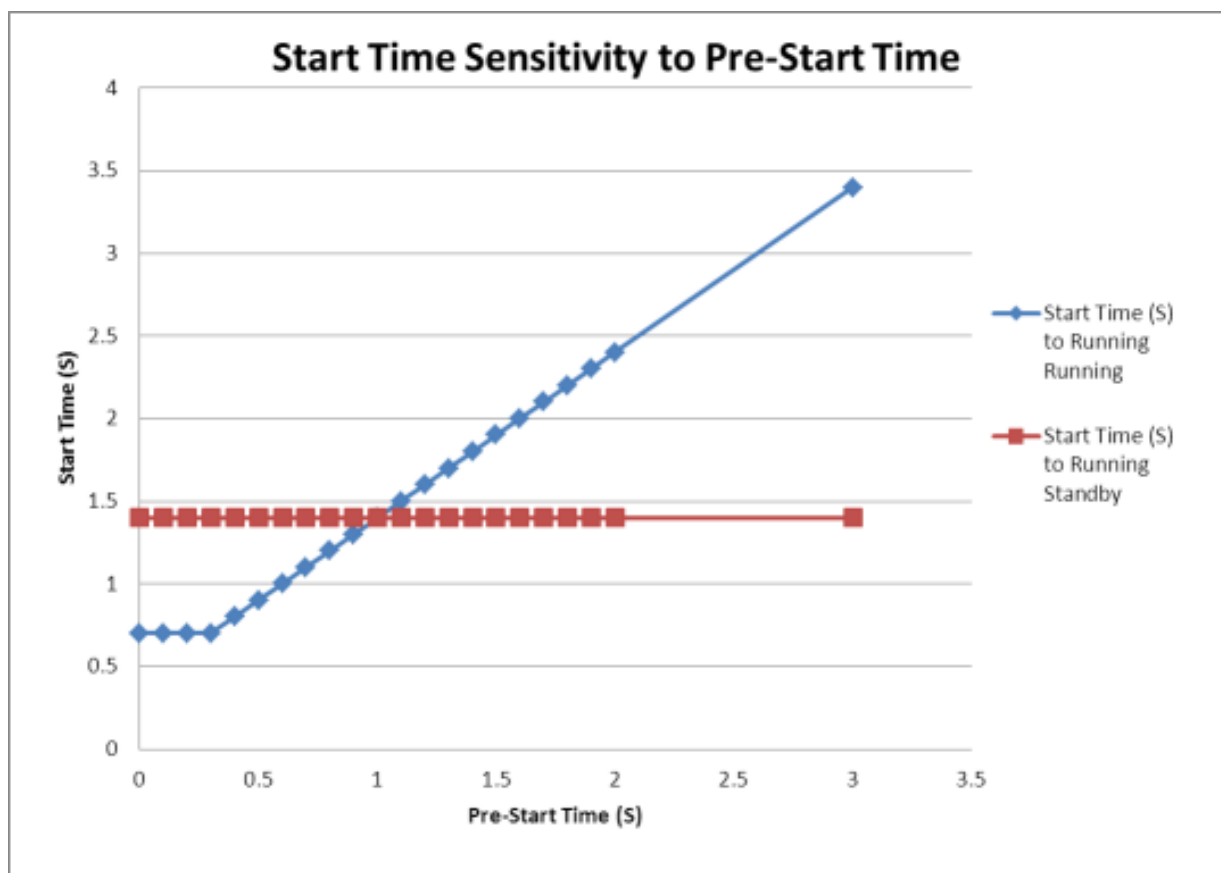


Figure 6.14 Parameter variation test result

This testing provided insights into the behaviour of the system, particularly the tolerable levels of variation of these parameters before affecting the behaviour of the system. For example it became evident that there was an interaction between certain parameters and it became necessary to do analysis while simultaneously varying 2 and even 3 factors. This analysis was containable in a model based context as the test and the processing of results were automated and by making use of techniques such as the Classification Tree Method. Figure 6.15 shows the results of a 3 factor analysis which highlighted a relationship between 2 configurable timing parameters which if calibrated wrongly, with the Prestart_Wait time equal or lower than the HV_Init_Wait time, would cause a default engine start even in conditions where the preferred HEV standby was possible.

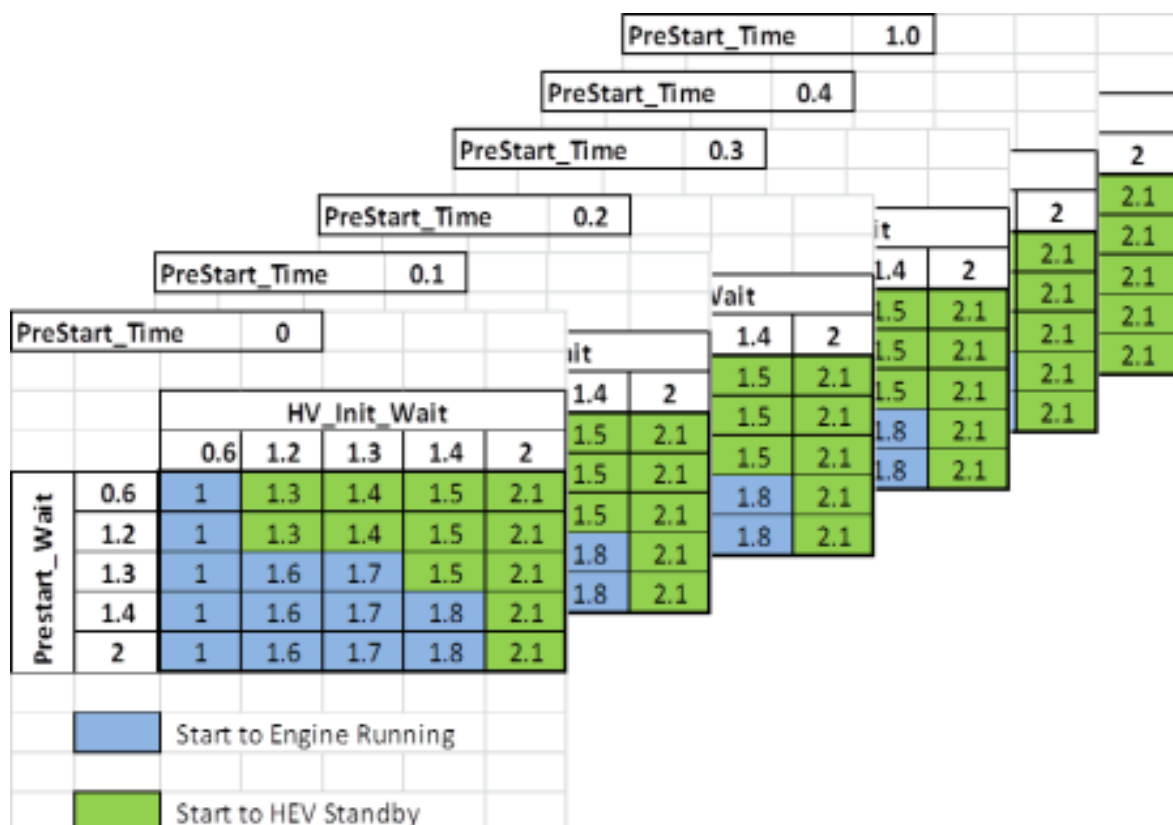


Figure 6.15 3 factor parameter variation test result

For some parameters a pseudo random test was performed. One example of this shown in Figure 6.16 was on the start button pressed signal to simulate multiple user presses of the start button, a likely scenario in a hybrid if the user is expecting the engine to start whereas the designed behaviour is for it not to start unless needed, or due to issues of switch de-bounce.

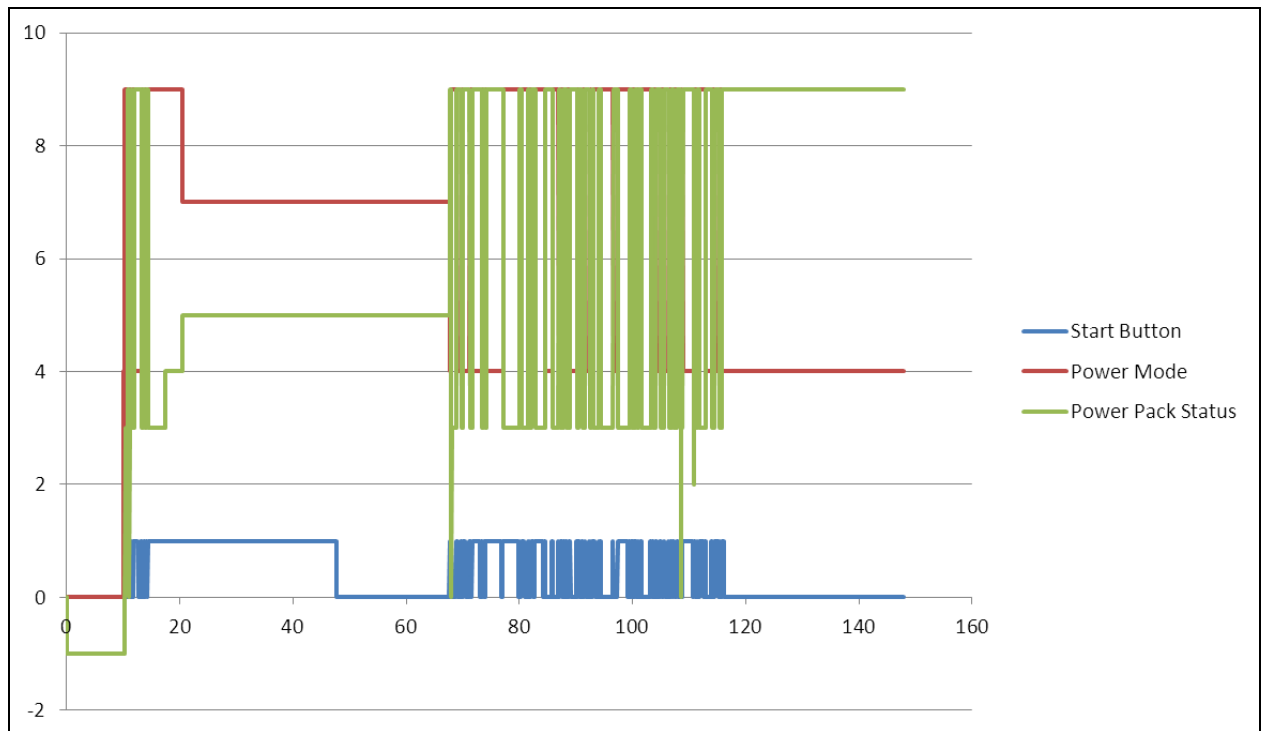


Figure 6.16 Psuedo random start switch test

The test results showed that due to the switch de-bounce timing and level of “inertia” within the system state transitions that the pseudo random input did not lead to undesirable behaviour e.g. rapid engine on off transitions, and always recovered to normal behaviour.

Fault Injection Testing

The next set of testing was based on fault injection. Studies of previous robustness issues had shown that during initialisation a key factor is not just that a fault has occurred but also *when* it occurs during the initialization phase that can determine whether it will lead to

failures. In cases where there is a critical time window in which the system is vulnerable if a specific abnormal event occurs it can be difficult to find and replicate at a physical level. Hence the fault injection testing looked at the impact of injecting faults, such as ECU resets which can occur due to supply voltage dips or watchdog resets, at various times during the initialisation sequence. This testing gave insight into the recoverability of failures during the initialisation. Figure 6.17 shows an example of a recoverability test for a drop out of the Body Control Module which controls the power mode. It shows that in this case that despite the drop out at 90.3 seconds the system re-initialised quickly enough to still detect the start button press and initialise to a ready state but with an increased start time. In situations where the ECU drop out occurred nearer the end of the switch button press there was insufficient time to initialise and detect so no start occurred however the subsequent press always resulted in a good start.

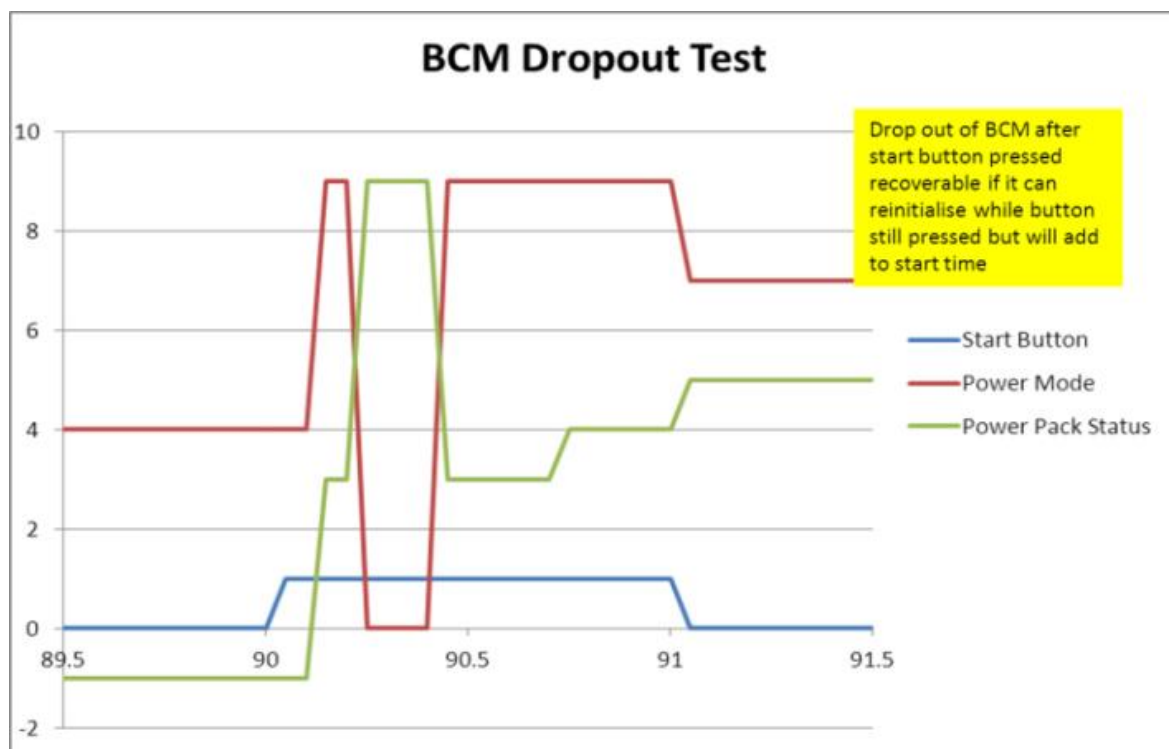


Figure 6.17 Fault injection test result

Correlation Testing

When representative physical systems become available it is advisable to do correlation tests with the modeled behaviour to give confidence in the results derived from the model.

For the hybrid system initialization CAN bus logs were taken to check and correlate the behaviour of the actual system against the model. Figure 6.18 illustrates a result from this showing that while there was good correlation in the behaviour between the model and the actual vehicle there were differences in the timing of the state changes.

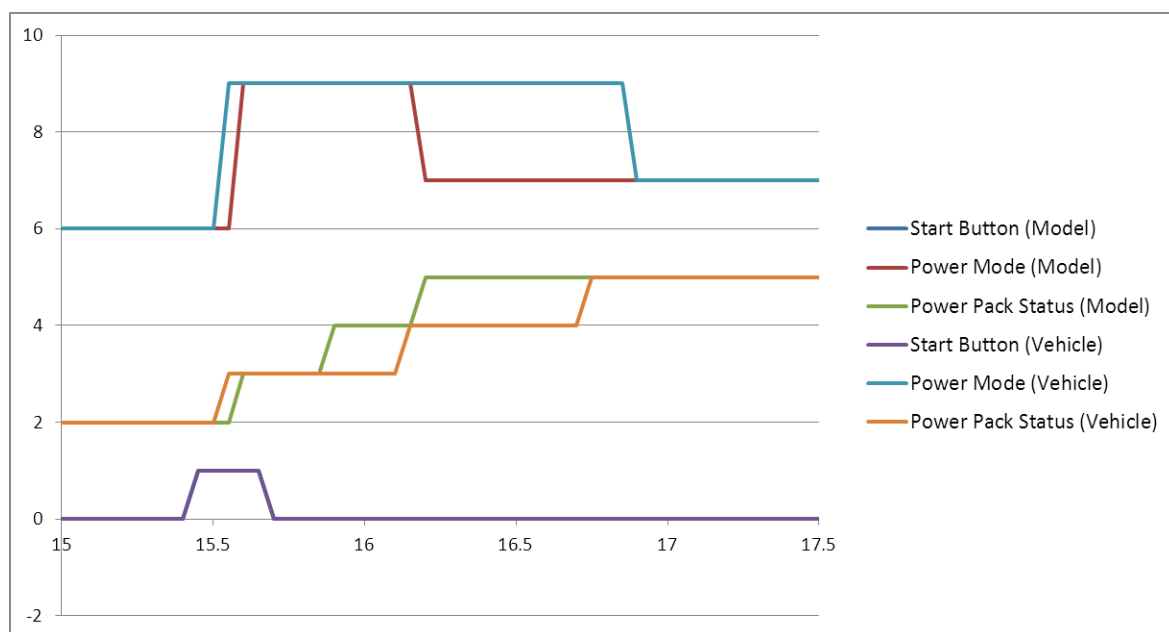


Figure 6.18 Initial correlation test result

Subsequent analysis found the major variation was in the pre-start time and crank time parameters which were both taking 600ms in the measured data as opposed to 300ms nominally in the model. Figure 6.19 shows a result from the test with the pre-start time and crank time parameters set to 600ms which exhibits a much closer correlation.

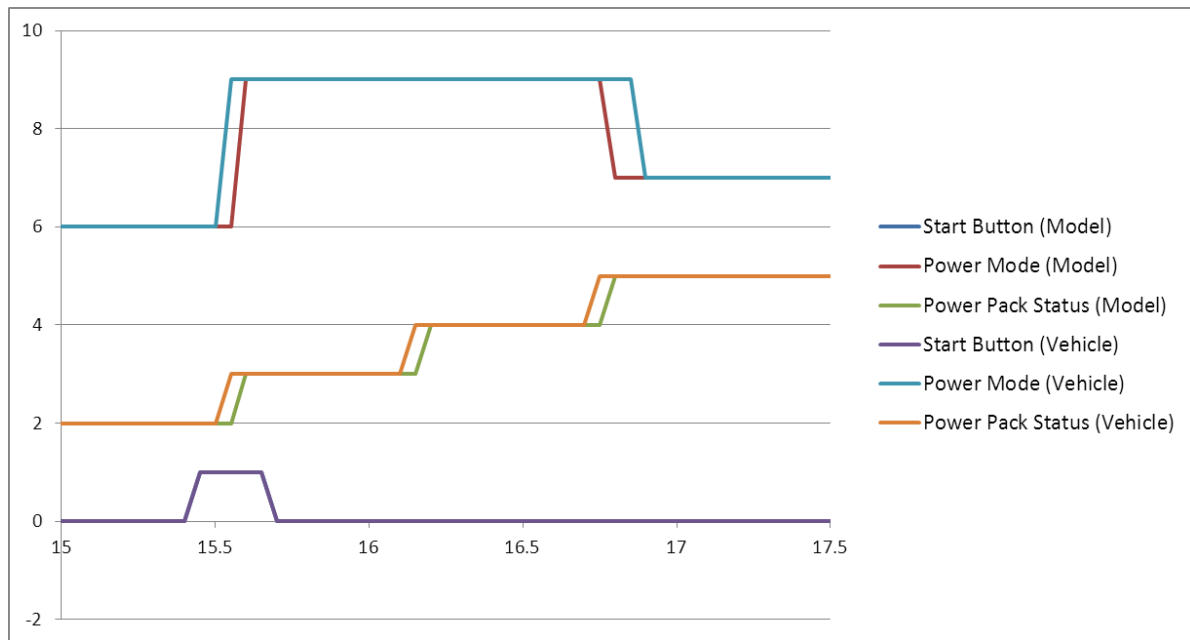


Figure 6.19 Updated correlation test result

The slight delay in the model to react to the start button press suggests the actual switch monitoring strategy is detecting the press faster than the specified nominal of 100ms. It can also be seen that there is a 200ms lag in the powermode reacting to the engine running powerpack status in the measured data at 16.7 seconds which is not exhibited by the model. This can be partly put down to the publishing delay in the powermode CAN signals which are sent on a 100ms frame rate with the residual being attributed to processing time for the incoming message. However these levels of variance are well within the bounds of the levels of variance tested and so give confidence in the modeled test results.

6.5 Discussion

In this section conclusions are given on the differences from the previous application and hence the transferability of the methods and also on utility of the individual techniques.

6.5.1 Robustness Cases

It was found the method of robustness arguments using GSN based diagrammatic arguments could be applied to the hybrid initialisation without requiring any changes to the generic methodology. It was possible to define a high level robustness goal and then conduct a high-level argument decomposition into modules based on organizational responsibilities. The further decomposition was possible using expert input and drawing on the findings from case studies.

The decomposed goals were different from the Infotainment application reflecting the different technical concerns of the applications. The Infotainment application was very focused on the communication and ensuring that the implementation was homogeneous – this having been found as a cause of issues on previous systems. The hybrid application was focused on ensuring solutions were in place to the typical issues with distributed systems using results from case studies, as no previous system of this type had been produced. The hybrid application represents a more general case of CAN communication rather than infotainment specific MOST communication so it is more likely that the argument modules can be re-used on other applications as well as future Hybrid systems.

The robustness cases were reviewed with 2 hybrid system specialists who were able to understand them. The robustness case was seen as useful in providing a common focus for various activities contributing to ensuring robustness making it visually reviewable for

assessing whether there are any omissions and as a mechanism for documenting and reusing good design practice.

The level of effort to develop the robustness case was 70 hours although the argument for component robustness was not developed as the work focused on the system design and verification argument modules.

6.5.2 Robustness Modelling

The generic method for robustness modelling was also found to be applicable to the Hybrid system initialisation case with the same generic modelling approach being able to be reused. For the Hybrid case the resulting model was significantly simpler than the Infotainment robustness model as shown by the metrics listed in Table 6.1

	Infotainment Robustness Model	Hybrid Robustness Model
ECUs	9	4
Simulink Blocks	520	231
State-machines	10	4
States	4610	159
Transitions	8381	261
Model Executable (MEX) file size (kB)	11539	755

Table 6.1 Comparison of model size between infotainment & hybrid robustness models

The Infotainment model was larger primarily because of the amount of complexity in each node to accurately represent the MOST communication control channel behaviour and secondly due to a higher number of ECU nodes. Again the Hybrid case is likely to be more typical of most vehicle systems which CAN networks and have less complex interactions.

The development and testing of the model identified a limited number of valid concerns which has either been detected by peer review or could be discharged as a result of mechanisms in place which avoided failures. It was also possible to demonstrate that the system behaviour was acceptable under external failure conditions such as drop-out of modules providing inputs.

The hybrid initialisation robustness model took 250 hours to develop, significantly less than the Infotainment model partly due to the reduced complexity of the application but also due to the ability to use established generic model structures.

6.5.3 Ability to Use Methods in Combination

The findings are that the two techniques are highly complementary and allow the exploration of robustness in the early design stages both as a structured thought experiment and through practical testing. The robustness argument will help identify critical design goals for robustness and the corresponding test requirements. By building an executable model of the robustness critical areas of the design, particularly the ability to function rather than the functionality itself, the design can be more completely understood. Through testing of the model the design can be checked for comprehensiveness and insights can be made regarding the underlying behaviour and the tolerable levels of variation understood.

In making these techniques tractable it was necessary to focus on particular areas of the system design. There will inevitably be trade-offs between breadth and depth which must be guided by experience and expertise but this should not be surprising as in any form of modelling endeavour skill is required. In doing this it is important to understand factors which present the most risk. In this case a subset of robustness factor was selected based on results from the case studies. Similarly the model has to be an abstraction of the full system and so inevitably will not exhibit behaviour attributed to un-modeled attributes. Again the decision

on what aspects of to be model must be made on basis of expertise and experience. Correlation testing can be conducted once physical properties are available to check that a limited set of behaviours corresponds to the model. Obviously this is after the event as the design has been implemented by this stage but it can give a level of confidence in the much wider set of testing that can be been done on the model and confidence in the use of similar models on future designs.

While additional up-front effort is required, more than 300 man hours in this instance, for these techniques early detection of robustness faults can greatly reduce workload later in development. In the next chapter the costs of the effort and potential benefits for the methods will be examined further.

6.6 Updates to Generic Methods

This section details where the generic methods can be extended as a result of the work on the hybrid evaluation study.

6.6.1 Robustness Cases

The hybrid case study reinforced the approach of decomposition of the robustness argument at a high level on the basis of organisational responsibilities, both within the organisation leading the development of the system and externally, to be able to cascade design requirements and allocate ownership appropriately. To make this division of responsibilities clear the use of colour coding to indicate ownership of achieving goals was introduced to the GSN diagrams.

The further break down of the argument structure becomes dependent on domain knowledge either directly from experts or through knowledge capture and application such as

design guidelines and case study reviews of previous issues. The argument can be used to set robustness goals, i.e. design requirements, for which part of the solution evidence is testing through the robustness modelling.

6.6.2 Modelling

6.6.2.1 Modified Structure of Generic Robustness Model

The generic approach to the modelling is fundamentally unchanged but enhancements have been made to support testing, communications between ECU blocks and to recognise that the system operating mode will be determined within an ECU. The modified structure of generic model is illustrated in Figure 6.20 and will be subsequently described.

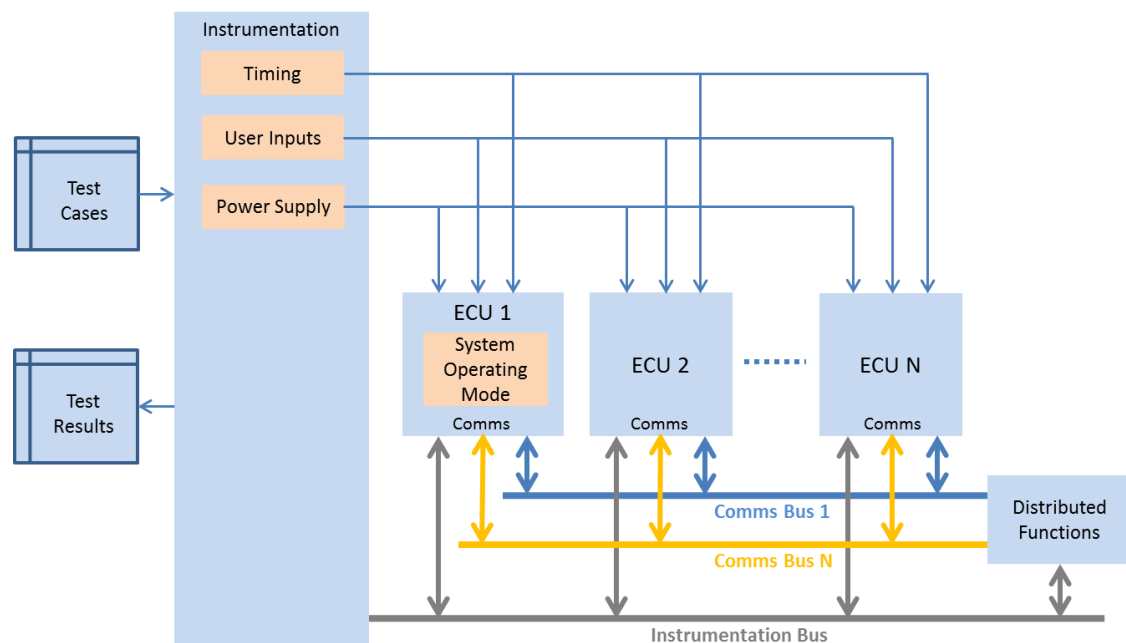


Figure 6.20 Modified structure of generic robustness model

The Instrumentation Block is used to execute test cases and record corresponding test results. The test cases are time series of events which correspond to potential activation patterns of failures including: user inputs, power supply behaviours, parameter variations and

potential fault behaviours of individual parts of the system. The instrumentation block stimulates the inputs of the wider model appropriately through: direct control of inputs, modification of parameters and activation of fault behaviours within individual ECUs via the Instrumentation Bus. This Instrumentation Bus is also used to capture the resulting behaviour of the individual parts of the model, e.g. through node status signals, which the instrumentation block records along with the input data as the test results. The test cases and results were generated and analysed using Microsoft Excel which was converted to and from Matlab files but this could be done through more bespoke test automation and data analysis tools as required.

The communication between the ECU blocks is implemented as a bus comprising individual signals rather than a set of discrete outputs. This makes the model more easily extendable as new ECU's only require to be connected to the bus rather than adding individual connections to each corresponding input. It also corresponds to the actual structure of automotive networks and it is possible to implement multiple busses corresponding to those on the vehicle. This becomes useful when looking at effects of failures within a bus or error propagation across the system of systems.

The block controlling the system operating mode (or "Powermode") was moved to be within an ECU rather than a standalone block. This reflects the reality of the implementation and in particular that the power mode signal availability is dependent on the state of the hosting ECU.

6.6.2.2 Plant Modelling

In the case where a plant model is required the following approach can be taken. The plant model should be only as complex as required to exercise each state and transition of the control model. This means that it can be a state based model itself which allows it to be

incorporated within the ECU state machine model alongside the control model. Continuous behaviours should be abstracted into states e.g. speed can be abstracted to: stationery, accelerating, constant speed and decelerating.

6.6.2.3 Testing Strategy

The approach for testing the robustness model can also be generically applied encompassing reachability, parameter, fault injection and correlation testing.

Reachability testing is to identify areas of the model, or the design it relates to, which are incomplete. It is analogous to MCDC (Modified Condition/Decision Coverage) testing of software which is intended to prove all entry, exit, statement and decision branches of the code can be invoked and that every condition of a decision can have an effect (Hayhurst et al.,2001). In a similar manner systematic testing is conducted to ensure that every state of the model and transition within the model can be reached.

Parameter Variation Testing is conducted to investigate the behaviour of the system resulting from variations in critical parameters, particularly the tolerable levels of variation of these parameters before resulting in failures. Initially single parameter sweeps of repeated tests with incremental variations in the value of a critical parameter are conducted to determine their individual impact. It may be necessary to repeat these in particular modes of the system as failures may only occur in these modes. There may be interactions between certain parameters and if this is suspected then analysis can be conducted while simultaneously varying 2 and even 3 factors exploiting test automation capabilities of model based testing. Pseudo random test variations can be performed to simulate user inputs or voltage drop-out behaviours to test for failures whereby the sequence of events is important.

Fault Injection Testing examines the impact of injecting known faults, such as ECU resets which can occur due to supply voltage dips or watchdog resets. These faults must be

selected based on previous experience and the local impact modelled, e.g. re-initialisation of ECU or loss of communications, along with the ability for the instrumentation block to control them. Testing needs to repeat the fault injection at varying times across different system states as there may be a specific time window of vulnerability in a specific mode for the fault to result in a failure.

When representative physical systems become available it is advisable to do correlation tests with the modeled behaviour to give confidence in the wider set of results derived from the model.

6.6.3 Interaction Between Methods

As illustrated in Figure 6.21 there can be useful synergies between robustness arguments and robustness modelling. The robustness argument can provide design requirements which are embodied in the robustness models as solutions to issues known to cause robustness failures. The solutions that are modelled provide design evidence for the robustness argument. The robustness models themselves are then used as test artefacts for model based robustness testing. The robustness argument can provide the verification requirements for this model based testing and correspondingly the results of the testing provide evidence for the robustness case.

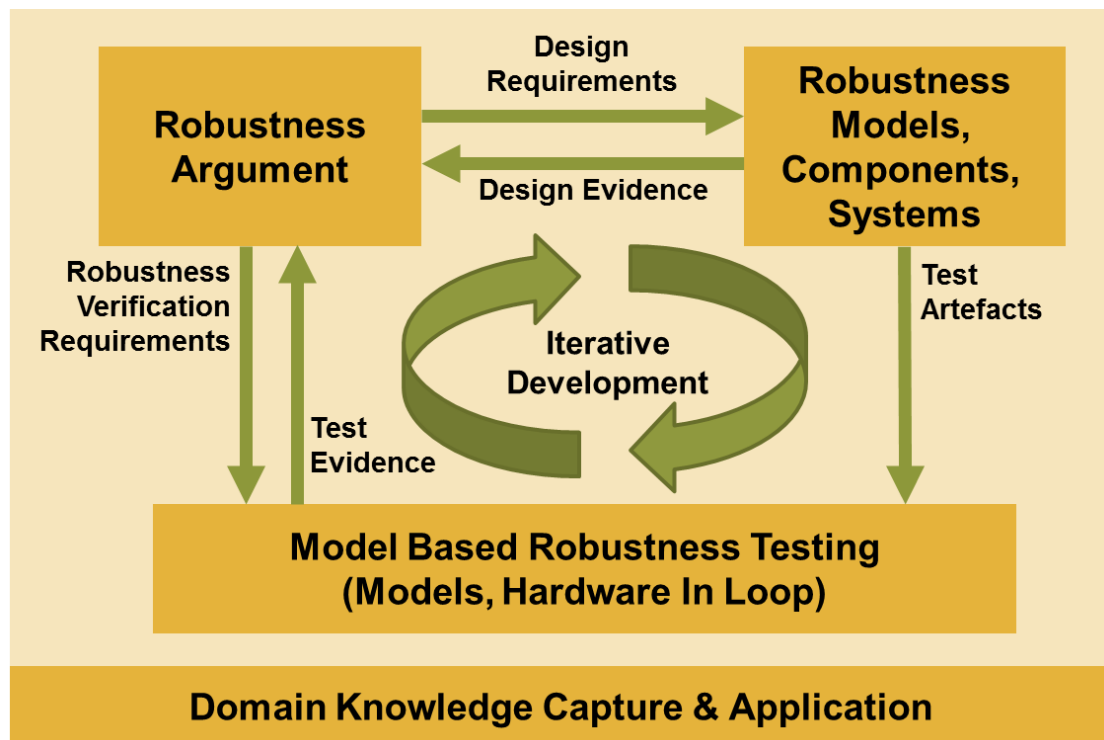


Figure 6.21 Interaction between robustness case and robustness model

This approach can be applied in an iterative fashion as a design matures from model to components and systems through the use of hardware in the loop techniques. However to be effective the technique must be underpinned by capture and application of domain knowledge, specifically knowledge of issues leading to robustness failures.

Chapter 7 – Discussion

7.1 Introduction

This chapter discusses the degree to which the work of this thesis answers the high level research question of whether a viable framework of methods to substantially improve robustness in the design of complex automotive electronics systems can be developed. Key discussion points concerning the viability of the methods proposed are; whether they are tackling the right problems, if so are they effective in doing this, are they efficient and likely to provide a cost benefit and how would such methods be practically deployed? Finally the novelty of the methods is discussed and specific contributions to knowledge identified.

7.2 Targeting of Methods

An overarching concern is whether the methods are addressing areas which are the root cause of robustness issues. The literature review demonstrated the scale and seriousness of issues concerning distributed automotive electronics, for example through the statements made by senior industry figures and the evidence of recalls quoted in Section 2.4.2. However the literature review noted an understandable reticence to put specific details of the nature of such problems into the public domain. This led to a requirement for the case study investigation in Chapter 3 confirming the nature of robustness issues to ensure the methods would be targeted appropriately. This study found that of the 43 faults analysed 26 (60%) were robustness related issues due to faults external to the system controlling service delivery, and hence it was concluded that robustness failures are a key issue to be addressed. The case study identified 73% of robustness faults as transient faults, created in the development stage, originating outside the system, affecting software. The primary cause

of this type of fault was communication errors, the most significant of which is the timing of interactions (race conditions, early and late interactions) followed by erroneous or missing input signals, spurious communications or corrupted messages. In a significant number of cases the precipitating cause for such communication errors was shut-down and re-initialisation of external systems, often due to low voltage transients. The case study identified the need for top down design approaches and also early validation approaches recognising the need for behavioural rather than just structural models in order to exhibit emergent behaviour. These findings were the basis for the development of the framework for design for robustness including the specific new methodologies of robustness cases and robustness modelling. As the case studies were all from a single source OEM then it is unknown whether the causes of robustness issues would be equally applicable to other OEMs, however it is known that the underlying technologies, e.g. CAN, and suppliers are common.

Further confidence in the appropriateness of the targeting of the methods can be derived from the infotainment and hybrid applications. These were selected as real concerns from industry that embody the types of issues found in the case studies rather than hypothetical ones. They also demonstrate the ability to use the methods across diverse automotive electronics applications rather than being specific to one type. Whether the methods are more universally applicable beyond automotive electronics has not been investigated as part of this work. Whereas this research draws on inputs beyond automotive further work could take the approaches subsequently developed in this thesis and test their application beyond automotive.

7.3 Effectiveness

Having confirmed that the methods are targeted towards the right problems then the next question to determine if they are viable is whether they are effective? To answer this it is necessary to examine the results from the pilot applications of the methods to infotainment and hybrid propulsion systems.

7.3.1 Effectiveness of Robustness Cases

Robustness Cases were developed using Goal Structuring Notation for an infotainment system and for the initialisation of a hybrid propulsion system in Chapters 5 and 6. From these studies a number of positive indicators of the effectiveness of the technique were observed.

Firstly the technique allows a broad and strategic approach by providing a structure through which it is possible to justify the approach to robustness at both an overall and detailed levels. In doing this it provides a structured approach which links many diverse activities, e.g. component specifications, system specifications, verification, supplier management standards and design processes, into a cohesive overall approach. Normal practice would have these elements in individual design and specification artefacts with a range of owners which the collective coverage would not be apparent. By having this overall view, gaps were identified in the applications investigated. Specific examples of this for the infotainment system were; a lack of system level verification methods and a need to get specific evidence of supplier resource capability. On the hybrid application the goal to specify adequate measures against known initialisation and shutdown failure modes highlighted both top down and bottom up gaps. From top down there were areas where it was not clear whether requirements were cascaded and solutions in place to issues such as flash

write robustness. From bottom up it made explicit many design goals e.g. *“design should include super-states which guarantee transitions out of sub-states for major events”* which were implicit in the actual design. However as these goals were not stated they would not be explicitly verified in the implementation of the design or could be transferred to other designs.

As the technique is hierarchical it allows the user to focus in on particular areas of concern in more detail based on previous experience. For example, from the case study review initialisation and shutdown were identified as high risk areas. In the infotainment robustness argument a specific robustness objective and argument module was added for initialisation and shutdown to focus on this area. In the hybrid propulsion study the whole argument focused on this one topic at a high-level of detail and was able to define the particular areas of investigation for the robustness modelling.

A strength of the robustness argument technique is that it can be used early in the design process. This allows necessary measures to be identified and incorporated into the design and development processes before too much effort and resources are committed. The technique was found to be useful in promoting a conscious discussion regarding robustness objectives and what they mean in practice. In the case of the hybrid initialisation what was the definition of *“acceptably robust”*. In Chapter 2 a definition of acceptably robust was derived from the definitions given by Avižienis et al (2004) of: an acceptably robust system is one whose behaviour in the presence of external faults correctly implements specified services to the user which may be a limited subset defined in a degraded mode. This resulted in specific goals being stated for the system behaviour in the presence of faults e.g. *“PCM brownout should result in re-initialisation of hybrid system”*.

The use of goal structuring notation as a diagrammatic technique was found to be effective for communication and review, with domain experts in both the infotainment and

hybrid areas, even by those initially not familiar with the notation. The GSN diagrams were not reviewed by anyone without expertise in the domain, such as higher level managers, but it was expressed by the technical experts that they believed that existence of a structure which showed an overall strategic approach would be useful in being able to argue that they had taken sufficient measures. In taking an approach to argument decomposition which reflected organizational responsibilities it helped clarify and make explicit roles and responsibilities, particularly between supplier and the OEM in the case of the infotainment system and between different teams within the OEM in both cases.

It was also noted in two specific areas that the method promoted ensuring evidence is actively and systematically collected and reviewed rather than assumed. The first was for legacy components that may be performing below their assumed capability which becomes critical within the new system context. In the case of the infotainment system there were carry-over components from previous systems which the goals required a review of previous test results. The second area was that of supplier resource capability which was generally assumed but to satisfy the goals required specific evidence of individual capability.

A limitation is that the technique is highly reliant on expert knowledge to be effective in managing the trade-off between coverage and depth in determining the breakdown of robustness objectives and the level of detail required in a particular area. However from the literature review it was concluded that domain knowledge is essential to complex systems and Goal Structuring Notation robustness cases offers a structured and hierarchical method of applying and capturing this. The method provides a means of integrating domain knowledge from several experts providing individual argument modules in a format that allows peer review at an early stage in the design process. These argument modules have the potential to become reusable examples of best practice or design patterns and hence offer a means of sharing domain expert knowledge. This would require repeated application to confirm the

extent of the re-usability, which is suggested as future work, but in setting design goals and identifying solutions to known issues at a minimum provides example solutions for reference. The biggest opportunity for re-use is where the goals are expressed at a level which is not specific to the particular application, e.g. *“each ECU should have specified fault response for the loss of each incoming signal”*.

It is important to note that in this context modularity applies to the argument not necessarily the system under development which typically for a system of systems will be heterogeneous in nature. However the arguments can be used to support the application of modular systems, such as in the infotainment case study, through identifying key supporting objectives such as composability and solutions such as proof of adherence to interface standards.

While the robustness case gives a context for a structured thought experiment it does not specifically address the issue in some of the cases studied of unforeseen use cases. A structured approach to elicitation and analysis of use cases whereby the user deviates from intended operation of the system is an area for further work.

7.3.2 Effectiveness of Robustness Modelling

Robustness models were developed for the initialisation of a fiber optic network based infotainment system and for the initialisation of a hybrid propulsion system in Chapters 5 and 6 and a generic robustness modelling method proposed. From these the following observations on the effectiveness of the robustness modelling have been derived.

Firstly the models did offer a new capability to test and optimise the robustness of the systems not possible with physical components. In the case of infotainment system the impact of changing design parameters on the system initialisation and the effectiveness of different strategies was able to be assessed via the model. In doing this it highlighted

emergent behaviours of the system which were not readily appreciated. It was also possible to do a level of grey box testing not possible with physical parts, for example being able to observe the utilization of internal buffers. For the infotainment system much of the detailed functionality was set by external standards and the creation of the model led to a deeper understanding of these.

In the case of the hybrid system initialisation the construction of the model detected errors in the specification. Although in this case these were also identified by design reviews this shows how constructing a model can reduce reliance on manual techniques for early error detection. The dynamic testing of the model found examples of redundant states and race conditions, showing effectiveness of the modelling and the testing regime identified in the robustness case.

The model also gave a capability to test across a range of variability not possible with physical components, whereby only a specific instantiation of the system could be tested at any one time and only once parts were available. From the results of the testing it was possible to determine the tolerable range of variability of component performance for the system to meet its design objectives.

The generic structure for robustness modelling shown in Chapter 5 and 6 gives a basis for a model of a networked system of systems but a limitation of robustness modelling, as with any model, is that it can only give insights on the attributes that are modeled. It is dependent on domain knowledge in determining what detailed aspects of the system are modeled and which use cases are tested. It can be seen from the case study that this can be derived from systematic analysis of existing information. In particular the method used for analysis within the case study e.g. robustness fault pathology, classification scheme of faults and collection of activation path parameters, can be re-used on other data sources of issues seen within other domains. The case study analysis in this thesis identified the most prevalent

issues and hence these are captured in the generic model structure. The use of robustness arguments and P-Diagrams are suggested methods for further exploration of which application specific attributes should be modelled.

7.4 Effectiveness of Overall Design for Robustness Framework

It was also noted in the hybrid propulsion evaluation study that the robustness case provided an effective structured mechanism for determining the focus of the modelling so the two techniques are synergistic. In fact this test case included all elements of the proposed framework for design for robustness. Design principles from the literature review and domain knowledge helped to construct the robustness case, then a robustness model was iteratively developed and tested encompassing all stages of the framework. Although limited in scope some confidence can be derived from this that the elements of the overall framework combine effectively. As with any framework it is possible to tailor it to the specific application thereby helping to augment and reinforce existing practices.

7.5 Cost vs. Benefit

Although the case studies offer evidence for the effectiveness of the methods for design for robustness of complex automotive electronics a further question is if the likely benefits outweigh the costs of incremental effort involved in these techniques? To answer this firstly the cost of applying the methods must be considered and secondly the financial benefits from their use must be considered.

The infotainment robustness case took approximately 100 hours to develop and the hybrid propulsion initialisation robustness case approximately 70 hours. The tools used were freely available adaptations of generic tools, e.g. Microsoft Visio, hence the only costs were

man power related. The current salary for an automotive engineer in the UK is typically £35,000-£50,000 per annum based on currently advertised roles (Jaguar-Land-Rover, 2013). Taking the top end of this and assuming 1600 man-hours per year and a 100% overhead rate gives a total employment cost per hour of approximately £62.50. From this figure and the time taken to develop the robustness cases it can be seen that they can be constructed for a cost of less than £10,000.

The robustness models took considerably longer to develop and test, approximately 800 hours for the infotainment model and 250 hours for the hybrid initialisation model. Again these models used generic tools, Matlab Simulink and Stateflow, which are available with no specific outlay. Hence the cost for constructing and testing the models can be estimated as £50,000 for the Infotainment robustness model and £16,000 for the hybrid propulsion system robustness model. The higher cost of infotainment model is due to a combination of higher model complexity and learning effects.

In assessing the benefits of the methods it is more difficult to be precise as to their monetary value but the following analysis gives indicative scale based on conservative assumptions. These assumptions are based on the author's experience as a senior manager in an OEM with teams responsible for the activities described. In general the benefits of the methods can be considered to be prevention of robustness issues at design stage for robustness cases and earlier detection of latent faults due to emergent behaviour through robustness modelling. The benefit of doing this will depend on the particular stage of development reached before an issue is detected and will escalate as design progresses and goes into production. This will be illustrated by considering the case of an issue which can be fixed by a simple software change to a single part of a system e.g. a modification to the conditions for a single state transition. An overview of the calculation approach showing the

different issue resolution activities and relative levels of effort across lifecycle steps is shown in Figure 7.1 and subsequently described.

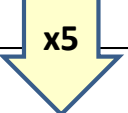
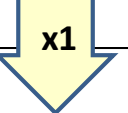
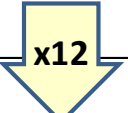
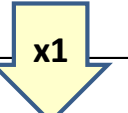
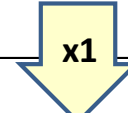
Life Cycle Stage	Resolution Activity for Software Issue			
	Fault Diagnosis	Update & Integrate Software Module	Re-Validation & Regression Testing	In-Market Deployment of Software
Initial Design	Not Required	Not Required	Not Required	Not Required
Prototype Development	Man Power Cost	Man Power Cost	Not Required	Not Required
Final Verification	 Man Power Cost	 Man Power Cost	Man Power Cost	Not Required
Production	 Man Power Cost	 Man Power Cost	 Man Power Cost	Software Update Cost Per Vehicle

Figure 7.1 Resolution activities and relative levels of effort across lifecycle steps

If this issue is eliminated by design approach, e.g. as a result of robustness case or by robustness modelling, then there is no cost. If this is found after the initial software is produced then there is a cost for diagnosis of the issue, say one man-day, and re-writing and re-integration of the software, say 10 man-days giving a total of 11 man-days at a cost of £5500 using the man power rates derived previously (88 hours at £62.50 per hour). These costs are based on the internal man-power rates, if the changes are implemented by supplier the costs could be considerably higher. It should be noted that even at this stage these may

not be the only costs associated with an issue, as it could result in lost time for testing or mask the finding of other issues leading to project delays.

If the issue is discovered during the final testing phase then there is a cost for diagnosis, say 5 man days, as having not been uncovered earlier diagnosis will be more difficult and a cost to re-write and re-integrate the software of 10 days as before. More significantly there is a cost for revalidation not just for the functions affected to prove the fix but additionally a cost for regression testing interfacing functions to prove they are unaffected by the change. This could be more than 50 man-days of effort for a system of reasonable complexity. Hence the total effort for finding an issue during final testing can be around 65 man-days even for a simple fix giving rise to a cost of £32,500 (520 man-hours at £62.50 per hour). Again there can be knock on effects that would result in much higher costs, for example if the issue caused delays in bringing a new or updated product to market.

In the worst case, the issue is not discovered until the vehicle is in production with significant numbers in the hands of customers. For a problem to get this far would suggest a low incidence issue which occurs under a very particular set of circumstances typical of the issues investigated in Chapter 3. Such issues by their nature can require significant efforts to diagnose and so an estimate of 60 man days is not unreasonable. The costs to re-write, re-integrate and revalidate would be similar to previous case giving another 60 man-days effort. This puts the cost of putting the fix in place at £60,000 (120 man-days giving 960 man-hours at £62.50 per hour).

The other part of the direct cost at this stage is the cost of deploying the fix which is dependent on; the number of vehicles produced with the issue, whether issue is to be fixed at routine service or as a special service action, and the means for deploying the fix. To address the latter first, typically a software fix as under consideration in this example is done using diagnostic equipment to reload new software to a particular ECU via the vehicle's diagnostic

interface and the vehicle network. Due to restrictions on bandwidth of the on-board network this can take a significant period of time for which the dealer can reclaim a cost from the vehicle manufacturer. For this analysis the cost for the in-market software update is assumed at £50 per vehicle based on 0.5 hours work at a rate of £100 per hour dealer charge rate (a Sept 2011 Survey by Warranty Direct puts the average franchised dealer labour rate at £95.83 in UK). Hence the variable cost for deploying the fix is £50,000 per 1000 vehicles in addition to the £60,000 fixed cost for putting the fix in place. Therefore if the issue impacts 100,000 vehicles then the cost of fixing it could be in excess of £5M even if this is done as part of a scheduled service.

Figure 7.2 illustrates the relative cost of implementing the proposed robust design methods against the potential cost per issue. This shows that the methods become viable at the point where they can detect *a single issue* which would otherwise have been undetected until final testing. For robustness issues which are only present during particular combinations of events and occur sporadically with low occurrence rates it is not unreasonable to assume they may not be detected before this stage due to low numbers of prototypes produced and the masking effects of other more prevalent issues. In the case of the methods of preventing or detecting an issue which otherwise could reach production, then they become highly viable from a cost benefit perspective with the cost of implementing them being a fraction of the potential cost of a single issue.

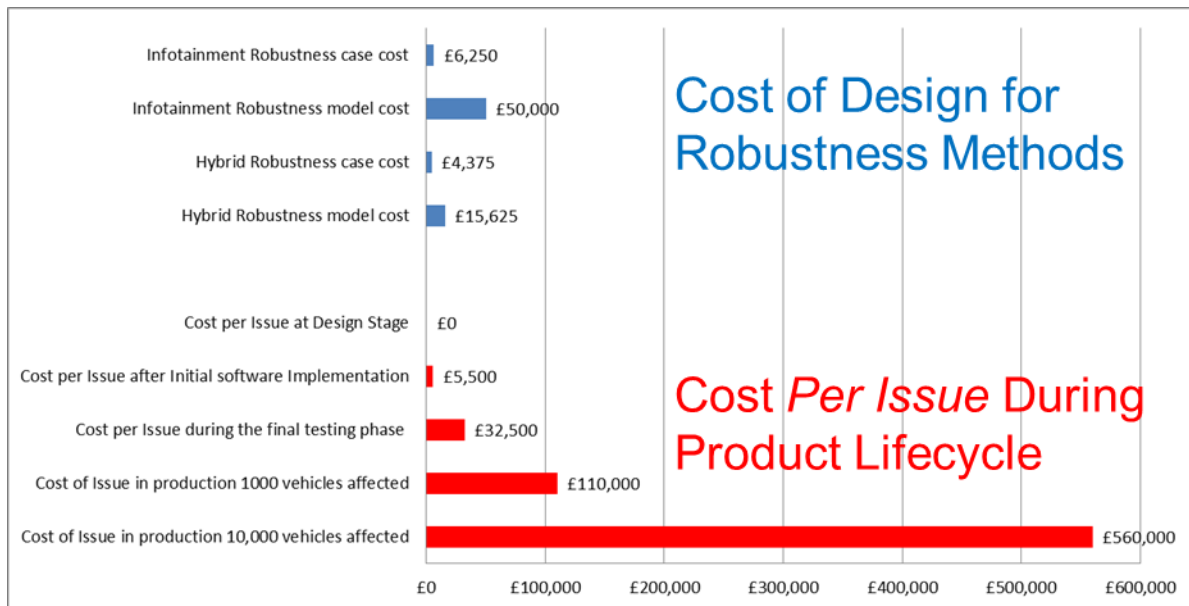


Figure 7.2 Cost of implementing robust design methods vs. potential cost per issue

While the direct cost of robustness issues are significant there are even more serious implications for loss of brand loyalty and reputation. J.D.Power are a company who conduct the largest surveys into the effects of failures experienced by customers on their subsequent purchasing behaviour. In their 2013 Vehicle Dependability Survey (J.D.Power, 2013) they found that when experiencing three or more problems with their vehicle, loyalty among owners of premium models declines from 55% to 39%. Clearly if 16% of customers are lost due to poor reliability then this has a major impact in terms of lost profit. This underlines the thesis's contention that robustness of non-safety critical functions is a business critical issue which needs more rigorous design methods.

7.6 Deployment of Methods

Having discussed the viability of the proposed methods and the framework for design for robustness, the next point of discussion is the practicality of their deployment. The two major prerequisites are availability of the necessary tools and of resource at a systems level.

As previously discussed the underlying tools, Microsoft Visio and Matlab Simulink and Stateflow, are widely available in large engineering organisations and used in other areas of design. However the availability of resource could be a constraint in the author's experience as a senior manager in automotive product development.

This constraint of availability of resource would suggest that an incremental approach to the deployment of methods, focusing on known areas of risk, would be the most viable introduction method. This would be similar to the pilot studies undertaken in this thesis but with a view to expand the knowledge of the techniques within the target area of deployment. By focusing on known areas of risk the potential benefits will be higher and as the benefit of the techniques become established then decisions can be taken on their wider use. If the techniques are successful then they should result in a net reduction of workload due to using less resource to prevent or detect an issue early in the design phase than to diagnose and fix later in the development phase or in production. However careful recording of appropriate metrics is required to substantiate this cause and effect.

Another enabler to the deployment of the framework is its ability to integrate with existing processes or new processes which are already being adopted. For complex systems a suggested approach by the International Council Of System Engineers (INCOSE) is that of Model Based System Engineering (Forder, 2012). In Model Based System Engineering (MBSE) models form the primary source of data for analysis and communication, as shown in Figure 7.3, and as can be seen from examples studied in the literature review is being adopted by many organisations.

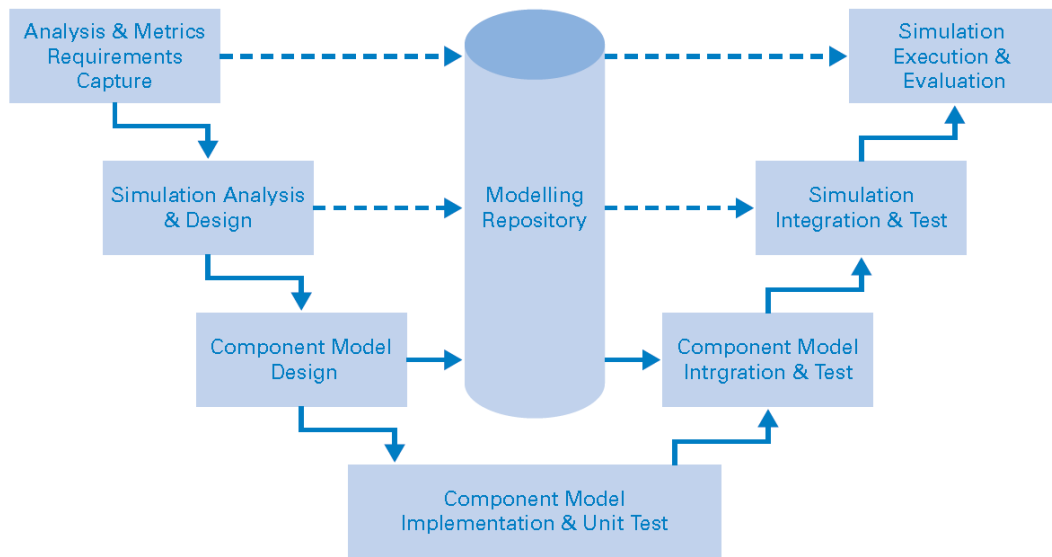


Figure 7.3 Model Based System Engineering from Forder, 2012

The approach developed in this thesis is entirely consistent with this Model Based System Engineering approach. This can be readily seen from Figure 7.4 which summarises the approach proposed, based on the simplified framework for design for robustness developed in Chapter 6 which exploits the synergy between the methods.

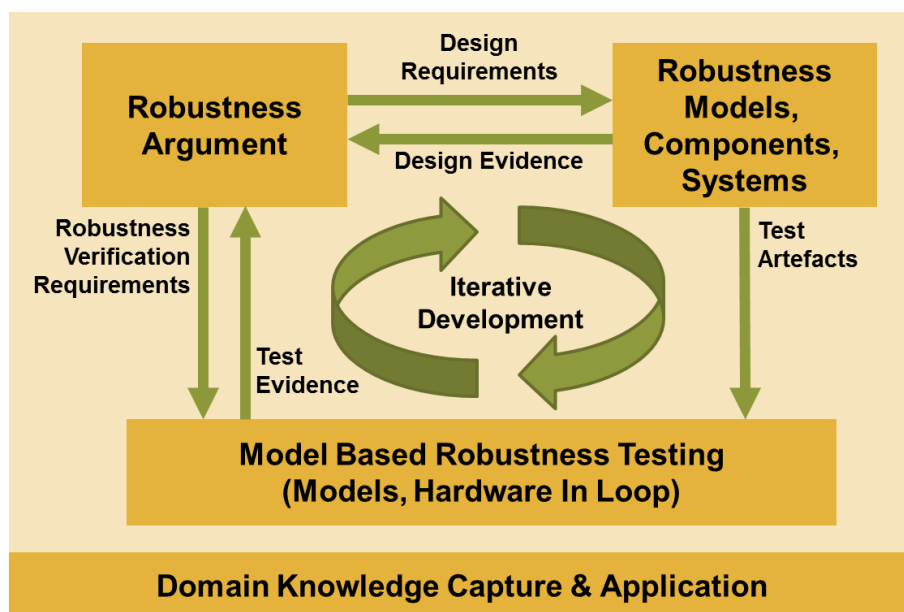


Figure 7.4 Simplified framework for design for robustness

One potential mechanism to provide sufficient focus for the roll out of the methods would be to have robustness specialists similar to functional safety specialists. The role of the robustness specialist initially would be to; develop robustness cases and robustness models, conduct robustness testing, support issue investigation and capture domain knowledge. As the techniques roll-out then the robustness specialist role may be more focused on training and coaching other engineers and the further development and dissemination of methods including their deployment as internal standards.

7.7 Contributions to Knowledge

From the literature review the key opportunities for contributions to knowledge in the field of robust design of automotive electronics were identified as follows:

- 1 Development of domain knowledge of impact in terms of prevalence, dependability characteristics impacted and the causes of robustness related failures in the area of automotive electronics.
- 2 Development of a design for robustness framework for complex automotive electronic systems. This should leverage best practices identified during the literature review but should include the following two areas identified for further new work.
- 3 Adaptation of safety cases to robustness to enable structured arguments for assuring the robustness of a system.
- 4 Development of an approach to robustness modelling based on understanding what are important factors to model pertaining to the robustness of automotive electronics.

These will be discussed in the remaining part of this chapter to confirm the specific contributions to knowledge that have been realized, including limitations, and finally to

discuss key learning at an overall level regarding the importance of domain knowledge in dealing with complex systems.

7.7.1 Domain Knowledge of the Prevalence and Causes of Robustness Related Failures in the Area of Automotive Electronics

The literature review identified that while there was evidence for the scale of issues in automotive electronics no systematic analysis of the prevalence and root causes of robustness issues was published. The analysis of a significant number of case studies in Chapter 3 has made a contribution to knowledge confirming the significance of robustness related failures is an issue and providing domain knowledge of significant root causes which has provided a focus for the development of methods. An overview of these findings has been included in a number of papers and presented at conferences (McMurran and Jones 2010, McMurran and Jones 2013).

Limitations of this study were noted due to the case studies coming from a single manufacturer and the analysis into the classifications only by the author. Suggested future work is to use the analysis method on new case data by more than one person to confirm the repeatability and reproducibility of the results.

7.7.2 Design For Robustness Framework for Complex Automotive Electronic Systems

The literature review identified that while techniques for robust design existed in the mechanical domain, e.g. Design For Six Sigma, they were not transferable to the electronics and software domains. Where potentially applicable methods existed in automotive electronics and software domains these had been mainly focused on safety critical applications and no overall framework for an approach to robustness existed.

Requirements for such a framework were based on a set of requirements defined using literature review and case study results. The framework developed in Chapter 4 to meet these requirements and subsequently refined which brings together a number of complementary techniques in a cohesive manner to address robustness.

While the elements of this framework were used in the Hybrid initialisation evaluation study the focus was primarily on the 2 new methods. Further application studies of the framework as a whole are required over a longer time period, allowing for repeated application to account for learning effects, to give higher degree of confidence in its overall effectiveness.

7.7.3 Robustness Cases

The generic method for robustness cases of using a hierarchically structured modular argument presented in a diagrammatic manner developed in Chapters 5 is based on that developed by Kelly and others for safety cases described in the literature review. What is novel is its application for arguing the robustness of a complex but not necessarily safety critical system, the general benefits being inherited from the original method.

The focus of the robustness argument is in particular justifying the freedom from service failures in the presence of errors due to external faults. Until now there has been no method for systematically reasoning and justifying why a complex system is robust that allows upfront risk analysis to determine where design effort to address systematic risks needs to be focused. To satisfy safety and robustness goals may require different approaches, e.g. a safety goal may require a system to fail into a safe state but a robustness goal may also require the system to recover.

7.7.4 Robustness Modelling

The literature review identified large bodies of work ongoing in the field of modelling complex systems, so is necessary to be explicit in defining what is novel in the approach developed within this thesis. The approach developed is unique in modelling an automotive complex system of systems at a level of abstraction which allows robustness properties to be viably examined. This approach is enabled by insights gained in the literature review and case study review which allow a sufficiently broad approach to deal with the problem at a system of systems level, while being a sufficiently detailed approach to model robustness relevant behaviours.

Current approaches decompose modelling into separate structural and behavioural models which are not amenable to analysis of system level robustness issues. From the literature review modelling of a system of systems is done at an architectural level showing structures, e.g. in SysML. Functional behaviour is typically modelled at an individual system level and communications modelled and analysed in isolation from functions on basis of meeting functional requirements such as maximum age of signals. However from the case study analysis of robustness issues it was clear that in order to detect emergent behaviour between systems it is necessary to do sufficiently detailed behavioural modelling across systems. The method of robustness modelling of automotive electrical systems developed in Chapters 5 and evaluated in Chapter 6 is made tractable by focusing on *the ability to function* rather than the customer functionality itself which is typically the focus of behavioural modelling. By focusing on the ability to function, utilising domain knowledge, it allows both detailed behaviour to be modelled but encompassing a number of interlinked systems.

Chapter 8 - Conclusions

In this chapter the main findings of each of the preceding chapters of this thesis are summarised followed by recommendations for future work.

8.1 Summary of Chapters

8.1.1 Summary of Chapter 1 - Introduction

In the introductory chapter the background to the problem domain addressed by this thesis was given. The increasing requirement for vehicle features results in a rapidly increasing complexity of automotive electronic systems. Automotive electronics exhibit properties of systems of systems including that of emergent behaviour and validation complexity. This brings with it major financial risks for automotive manufacturers due to field failures, launch delays, recalls and loss of customers. In order to mitigate these risks effective tools and techniques for the robust design of complex automotive electronic systems are required, but initial research suggests that little published work on robustness, as opposed to safety, has been done in this field.

Hence the research question identified was whether a viable framework of methods to substantially improve robustness in the design of complex automotive electronics systems can be developed. In order to address this research question the subsequent chapters of this thesis were structured as follows: Chapter 2 reviewed literature from both automotive other domains and highlighted where specific contributions to research could be made, Chapter 3 conducted a case study review of automotive electronic issues to understand the nature and root cause of robustness issues, Chapter 4 developed a framework approach incorporating the two proposed new methods, in Chapters 5 the method is developed through application to an

infotainment case study and a generic approach to the methods derived, in Chapter 6 these new methods were evaluated on a case study of a hybrid propulsion systems and in Chapter 7 the viability of the framework of methods developed and their contribution to knowledge was discussed. The findings from of these chapters are summarised in the following sections.

8.1.2 Summary of Chapter 2 – Literature Review

The literature review firstly sought to clarify what is meant by robustness especially as opposed to reliability. A robustness fault is an internal fault within a system that is instigated by an external fault causing an error to be propagated to the system. This is noting that the failure mechanism may be through triggering a dormant internal fault. An acceptably robust system is one whose behaviour in the presence of faults that successfully implements a specified subset of services to the user defined in a degraded mode. This is a distinct and different concern from reliability which is concerned with reducing the occurrence of internal faults in the first place.

The literature on complexity confirmed that complex systems are where they have interdependencies which make them inherently non-linear. These exhibit emergent behaviour due to difficulty in predicting this during design through human analysis or modelling. Systems of systems are a particular type of complex system whereby a number of individual systems are interlinked to share resources, e.g. information, power sources or hardware. These are particularly vulnerable to emergent behaviour due to the individual systems being designed in isolation with individual purposes, potentially at different times by different organisations and also due to their large scale.

Automotive Electronics are implemented as a system of systems, are highly interactive and hence highly complex. Additionally as they are embedded systems where there are time critical dynamic interactions they have a high degree of “*Dynamic Complexity*”. Review of

the literature also highlighted concerns with coping with this complexity coupled with a high rate of change and a short time to market. The approaches identified to address this were standards for open systems, sharing and reuse of software such as GENIVI and AUTOSAR as well as model based development. A key finding was that whereas there is evidence of the scale and seriousness of emergent issues in automotive electronics, such as the cost of recalls and statements by major industry figures, there is little published information on the nature of these problems.

The literature on a variety of potential approaches across industries to address the robustness of complex systems was then reviewed. It was found that robust design techniques such as Design for 6 Sigma are highly mechanically focused and have an emphasis on “right first time” which fundamentally misses the issue of emergence. In contrast “agile” approaches developed in the software domain are designed around an iterative approach which accepts that issues will occur which can be fixed at a subsequent iteration.

It was found that much of the research efforts and standards are focused on high integrity safety-critical systems which lead to solutions to guarantee high reliability as opposed to robustness. This primacy of reliability over robustness leads to solutions which are predicated on levels of redundancy which are not appropriate for non-safety critical systems on mass market products. However a body of good practice was identified for the design and architecture of complex systems embodying such concepts as simplicity, minimality, modularity, encapsulation, independence, composability, elementary interfaces and self-stabilization which can be applied with little or no piece cost effect.

Some of the approaches used in the development of safety critical systems were identified as being applicable to design for robustness such as the need to do upfront risk analysis to determine where design effort is required to address systematic risks. In particular

methods of diagrammatically developing hierarchically structured safety cases were seen to have potential to be adapted to design for robustness.

There is extensive work on modelling of complex systems ongoing which was reviewed but this highlighted the issue in resolving "big picture" architectural level modelling (e.g. SysML) with detailed "behavioural" modelling (e.g. in Matlab Simulink) for large scale systems. As none of the approaches were specifically focused on robustness the question of whether a complex system could be usefully modelled at a level of abstraction that allows robustness properties to be examined was unanswered.

Verification testing is an area where there is specifically robustness focused work published tackling the issues of how to practically test systems with extremely large state space. Formal verification methods were also reviewed and it was concluded that given their potential a framework for robust design should incorporate them as an approach. However formal methods tools and techniques need to mature and become lightweight through ongoing application research in high integrity automotive electronic applications. At a future stage the robustness models and knowledge of specific properties that need to be checked to prove robustness can provide a basis for formal analysis.

An observation running through much of the literature on complex systems is the importance of domain knowledge. Underlying this is the fact that as complex systems are not amenable to exhaustive analysis even by automated methods or full understanding by any single individual, then heuristic approaches based on domain knowledge to anticipating and finding issues are necessary. However as previously noted there is a lack of domain knowledge of automotive electronics robustness issues in published literature.

In conclusion the literature review identified key opportunities for contributions to knowledge as follows:

1. Development of domain knowledge of impact in terms of prevalence, dependability characteristics impacted and the causes of robustness related failures in the area of automotive electronics.
2. Development of a design for robustness framework for complex automotive electronic systems. This should leverage best practices identified during the literature review but should include the following two areas identified for further new work.
3. Adaptation of safety cases to robustness to enable structured arguments for assuring the robustness of a system.
4. Development of an approach to robustness modelling based on understanding what are important factors to model pertaining to the robustness of automotive electronics.

8.1.3 Summary of Chapter 3 - Case Study Review of Automotive Electronics

Robustness Issues

A review was conducted of 43 well-documented field issues in the area of automotive electronic systems. It was found that these were predominantly (60%) robustness related issues due to faults external to the system controlling service delivery, suggesting that current development techniques were less effective in addressing robustness and so supporting the need for improved techniques.

This does not necessarily imply that this portion of all failures are robustness related as the case studies didn't include problems that were fixed during development or in-service issues that had more minor impacts. However it does suggest that current techniques are generally picking up simpler failure modes but not more complex robustness related failures which then manifest themselves as issues when the vehicle is in service causing customer dissatisfaction. It therefore verifies the need for work in identifying improved techniques to prevent robustness related failures.

It was found that the primary impact of robustness failures are on reliability in 58% of issues studied, giving interrupted or degraded service, and on availability to provide the correct service in 38% of the issues studied. None of the issues were safety-related but any such issued may have been pre-filtered.

A fault classification scheme was used to identify the most common type of robustness fault as transient faults, created in the development stage, originating outside the system, affecting software. This accounted for 44% of all issues and 73% of the robustness issues, The primary cause of this type of fault was predominantly communication errors, the most significant of which is the timing of interactions (race conditions, early and late interactions) followed by erroneous or missing input signals, spurious communications or corrupted messages. In a significant number of cases the precipitating cause for such communication errors was shut-down and re-initialisation of external systems, often due to low voltage transients. These types of issues were identified as the focus of robustness methods to be developed.

Analysis of activation patterns of robustness failures concluded that while most were complex, involving 2 or 3 factors, a significant number (46%) only required a single factor or were permanent, i.e. if the fault occurred then so would the service delivery failure. Further analysis showed that in these cases either the activation factor had to be precisely controlled with respect to timing or level or it was combined with an elusive fault. Any robustness methods need cater for a variety of activation scenarios including at least 3 factors. Particular factors which were highlighted were user operations, low voltage transients and vehicle power-mode behaviour. While the study highlighted a number of important robustness related parameters a methodology for identifying and understanding these as part of the system design was identified as being desirable.

Design changes were made in response to the robustness issues in 88% of cases with these being predominantly software changes. Other common actions were specification changes (69%), verification method changes (62%) and FMEA updates (42%) which suggests these processes are not fully effective in preventing robustness issues. Hence it was identified that the design for robustness methods to be developed should seek to augment or improve these.

The analysis of the earliest detection points suggested some opportunity to “left shift” detection to earlier in the development cycle before vehicles tests, particularly if viable model based testing approaches can be developed.

8.1.4 Summary of Chapter 4 - Development of Design for Robustness Framework

Chapter 4 utilised the learning from the case studies of robustness issues to develop a set of requirements for a design for robustness framework before developing the framework using knowledge of potential design methods from the literature review. This framework is shown in Figure 8.1.

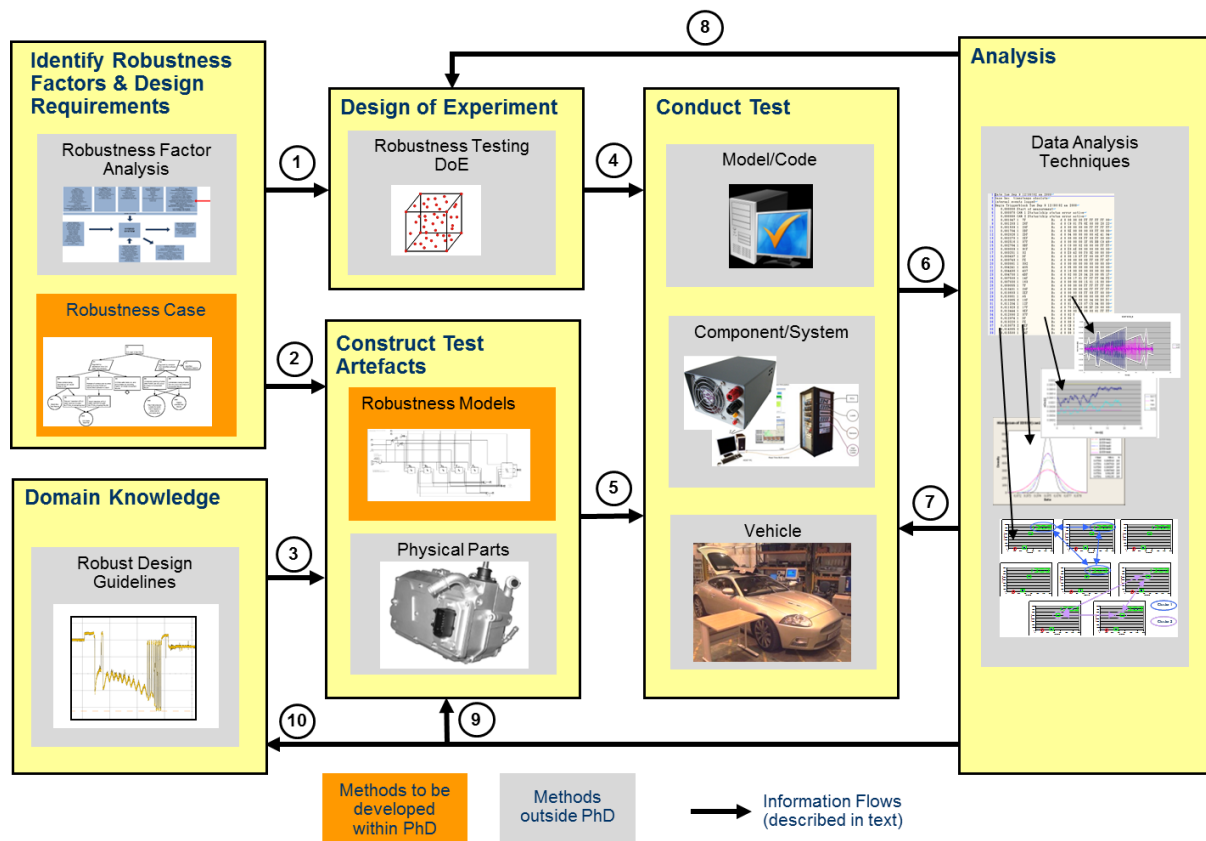


Figure 8.1. Framework for robust design of complex automotive electronics systems

The initial step in the framework is to identify key robustness parameters through variability analysis techniques such as review of previous design issues and parameter diagrams. At this stage a new technique is used of a “Robustness Case”. In a similar manner to a safety case the Robustness Case gives a structured argument for how robustness requirements are met. This serves as a structured thought experiment for the system designers making them consider and explicitly express robustness objectives at an early stage of the design. It also gives confidence to decision makers that there has been adequate consideration of robustness in the system design and there is traceable evidence of how robustness objectives have been met.

From the development of the Robustness Case two key outcomes are derived relating to the design and the testing of the system. Firstly the decomposition of the top level robustness

goal allows specific sub-goals to be derived for the system design which then has to provide appropriate solutions to these. Secondly detailed verification requirements are derived to provide the evidence of meeting the robustness sub-goals.

When designing the strategy of operation of a complex system, it is desirable to build an executable model of the relevant parts of the system and use the model to validate the robustness of the design before committing to its implementation. Hence the next step is to design and construct test artifacts at various levels of abstraction, which initially will be models of the operating strategy at a high level, before moving to more detailed models of the control and physical interactions and ultimately physical parts. In designing these test artifacts an input should be the design requirements identified in the robustness case as well as from robust design guidelines which should be embodied in standards, training and peer review processes.

The next stage is conducting tests on the design artifacts modelled, physical or combinations thereof. A major consideration for robustness testing is the design of experiments which is challenging due to the exponentially large state space of complex systems and the large number of experimental variables. Techniques for structured pseudo random testing are one approach which has been employed (Dhadyalla et al. 2010), but other applicable techniques include Classification Tree Method (Grochtmann, 1994) and covering arrays (Dhadyalla et al. 2013). It was also identified that formal methods, in particular model checkers can be used in future as these methods become mature and sufficiently lightweight.

Data analysis of the results of robustness testing can also be challenging. This is not just due to the volume of data but also because the test is not looking for the expected response to a stimulus as in functional testing but rather whether an unexpected response has occurred, which by definition is more difficult to find. Therefore, as with performing the tests, automation of the analysis and use of data-mining tools are important. Ideally

continuous measures of the health of a system and its components should be used as these can expose areas of the test space where levels of variation or interactions cause risk without manifesting in customer level failures which can be the subject of more focused investigation.

From the model based testing, which can be done at white box level on all modeled attributes, learning should be taken not just on the design of system but also what parameters could be made visible at a black box level which would enhance robustness testing of physical parts by giving a better view of component state and health.

8.1.5 Summary of Chapter 5 - Development of New Robustness Techniques through Application to an Infotainment System

In Chapter 5 the two novel techniques, specifically robustness cases and robustness modelling, were developed through application to a fiber-optic network based infotainment system. The chapter gave an overview of MOST (Media Oriented System Transport) which is a communication system with a flexible architecture designed specifically for automotive infotainment systems and used by many premium vehicle manufacturers.

A robustness case for a MOST infotainment system was developed based on the diagrammatic technique Goal Structuring Notation (GSN) with the top level goal defined as *“Infotainment system is acceptably robust in normal operation”*. The structure of the robustness case followed a modular approach based on principles identified in the literature review of; high cohesion and low coupling, supporting work division and contractual boundaries, supporting future expansion and isolating change. In practice this worked well being able to recognise OEM and supplier relationships and utilise existing design artefacts from a variety of sources. The technique added value not just in providing structure to existing measures e.g. specifications, statements of work, but also in highlighting areas where

additional measures needed to be taken. The diagrammatic notation was easily understood by domain experts, including those to whom it was new. A further benefit was that it provided a mechanism to ensure that evidence e.g. results of testing, is systematically gathered and collated. In complex systems, especially those with legacy components, it can easily but wrongly be assumed that testing has been done previously resulting in system failures. The robustness case was constructed in around 100 hours.

Taking the approach identified from the case studies of focusing robustness modelling on *the ability to function*, rather than the functions themselves, robustness models were constructed. These were initially of a generic system and subsequently a more detailed model of the behaviour of a nine node MOST infotainment system control channel during initialisation. The models were constructed using Matlab Simulink as a set of interlinked state machines representing the ECUs connected by signal buses representing the vehicle networks. This showed that it was possible to model the behaviour of the system to explore design alternatives to improve robustness which would not be economically viable with the physical system. The model exhibited emergent behaviours within the system which would have been unlikely to have been predicted through manual analysis. A further finding was that the model could enable parameters not normally observable on physical parts, e.g. memory buffer utilisation, to be studied. The infotainment robustness model took approximately 6 man months effort to develop but this included the learning effect of the development of the new method.

The methods which had been developed were then described as generic approaches to allow re-application on a wider basis. The process steps in developing modular robustness case were defined as:

Step 1. Define high level robustness goal,

Step 2. Define argument module decomposition,

- Step 3. Develop argument modules,
- Step 4. Review argument modules,
- Step 5. Review overall argument and
- Step 6. Correlate to design requirements.

The generic methodology for the development of robustness models was explained with reference to a generic model structure for which the elements were described.

8.1.6 Summary of Chapter 6 - Evaluation of New Robustness Techniques through Application to a Hybrid Propulsion Control System

To evaluate the proposed new techniques of robustness cases and robustness modelling, and prove their wider applicability, they were applied to the system initialisation of a hybrid propulsion control system. The initialisation of the hybrid propulsion system was chosen as an example of a simple high level requirement that requires complex system functionality to deliver involving several different sub-systems and was different to the infotainment application used to develop the methods in that the constituent parts are not homogeneous. This was important in showing the methods are more generically applicable.

The robustness argument helped identify critical design goals for robustness and the corresponding test requirements. A decomposition strategy broke down the argument into system design goals, component design goals and verification goals. The area of most focus was the system design which was decomposed into 10 sub-goals each of which were developed as argument modules which have the potential to be reusable examples of good practice although this would require follow-on studies to verify. Part of the argument for achieving the design goals was model based verification, hence the argument provided requirements for the system design and robustness model based verification.

By building an executable model of the robustness critical areas of the design, particularly the ability to provide customer level functionality rather than the functionality itself, the design could be more completely understood. In this case the model took approximately 250 man hours to develop and test. Through development and testing of the model the design was checked for comprehensiveness and insights were gained regarding the underlying behaviour and the tolerable levels of variation understood.

Four specific types of testing were undertaken as defined in the robustness case. Reachability tests of each state and transition, analogous to MCDC testing of software, was found effective in confirming completeness of design and model. Parameter variation testing of robustness critical parameters was conducted including two-way testing and pseudo random testing. Fault injection testing, particularly with respect to when the fault occurs, was used to confirm particular vulnerabilities at different stages within the initialisation, fault reactions and recovery mechanisms. Correlation tests were carried out when results from physical systems became available, which after calibration of 2 parameters showed a high degree of correlation between the behaviour of the model and the actual system.

The findings were that the two techniques are highly complementary and allow the exploration of robustness in the early design stages both as a structured thought experiment and through practical testing. There were some minor updates to the structure of the generic model to support testing, communications between ECU blocks and to recognise that the system operating mode will be determined within an ECU and is therefore subject to the availability of the underlying processing platform.

8.1.7 Summary of Chapter 7 – Discussion

Chapter 7 firstly discussed the viability of the methods developed in answer to the research question of whether a viable framework of methods to substantially improve

robustness in the design of complex automotive electronics systems can be developed. To ensure viability the cost of implementing the proposed methods must be less than the expected cost of failures detected. This required analysis of what the expected cost of a failure could be depending on when in the product lifecycle it was found. Secondly the specific contributions to knowledge arising out of this work were discussed.

To be viable of primary concern is whether robustness is an issue with automotive electronics systems and are the methods developed addressing the root causes of robustness issues? This concern was addressed through conducting the case study reviews which showed that the type of issues not found through normal verification were indeed predominantly robustness related and gave insights of critical factors which allowed the methods to be targeted appropriately.

Having confirmed the methods were well targeted, the next question was whether they are effective in prevention and detection of robustness issues? From the application of robustness cases to infotainment and hybrid propulsion systems a number of positive indicators of the effectiveness of the technique were observed. Firstly they brought a broad, structured and cohesive approach linking many diverse activities and by having this overall view gaps in approach were more readily apparent. The diagrammatic technique allowed the approach to robustness to be readily communicated and hence reviewed. Being hierarchical it also allowed particular areas of concern to be focused on in more detail and as the approach can be done early in the design phase it can identify areas of focus for subsequent design and verification activities. It was also noted that the method promoted a higher level of rigour in ensuring evidence is actively and systematically collected and reviewed rather than assumed. A limitation of the technique is that it is reliant on expert knowledge to ensure the robustness case addresses the right issues, however it does provide a mechanism for input and review by

range of expert stakeholders with the opportunity to systematically re-use best practice argument modules.

The robustness models gave a new capability to test and optimise systems not possible with physical components. They allowed design alternatives to improve robustness to be evaluated at an early stage in the design and in doing so allowed the completeness of design to be assessed and the detection of errors in specification. When executed they highlighted emergent properties of the system not readily appreciated from a static analysis of the design. The models also enabled testing across a range of variability not possible with physical parts to be able to determine the tolerable range of variability of component performance for the system to meet its design objectives. A limitation of robustness modelling as with any model is that it can only give insights on the attributes that are modelled which again relies on expert judgment.

Having gained confidence from the application projects that the methods will help to eliminate and detect robustness issues then an analysis was conducted of whether the likely benefits would justify the incremental costs of implementing the methods? This showed that the methods became viable at the point where they can detect *a single issue* which would otherwise have been undetected until final testing. For robustness issues, which are only present during particular combinations of events and occur sporadically with low occurrence rates, it is not unreasonable to assume they may not be detected before this stage. In the case of the methods of preventing or detecting an issue which otherwise could reach production, then they become highly viable from a cost benefit perspective with the cost of implementing them being a fraction of the potential cost of a rectifying single issue, even before taking into account potential loss of customers.

Deployment strategies for the methods were discussed proposing an incremental approach focusing on known areas of risk supported by robustness specialists. The approach

is consistent with model based systems engineering and simplified framework for deploying it this in this context was illustrated and is repeated below in Figure 8.2.

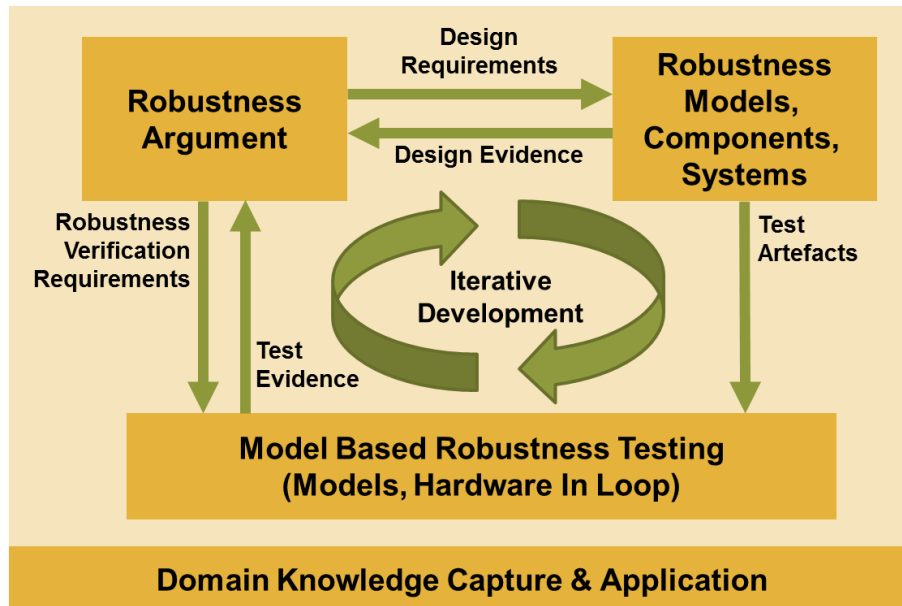


Figure 8.2 Simplified framework for design for robustness

Finally the contributions to knowledge were discussed. The first contribution is in the domain knowledge of the prevalence and causes of robustness related failures in the area of automotive electronics developed through the analysis of a significant number of case studies. The second contribution is the design for robustness framework for complex automotive electronic systems where previously no cohesive overall approach to robustness existed. The third contribution is in the adaption of methods for safety cases to robustness cases. The fourth contribution is in identifying an approach for modelling complex systems at a level of abstraction which allows robustness properties to be viably examined.

8.2

8.3 Future Work

The following sections outline specific areas identified for further work.

8.3.1 Domain knowledge capture, codification and dissemination of robustness issues

A major learning point from this work has been that it is possible to develop viable methods for the design for robustness of complex systems, however they have to be underpinned by domain knowledge.

This predominance of the importance of domain knowledge in dealing with complex systems is currently not reflected in the available literature containing domain knowledge, or as methods for systematically capturing and codify this knowledge or in specific training. This may be because such domain knowledge is important intellectual property in the form of know-how to give competitive advantage to an organisation or an individual working within that particular domain. A further factor is that of organisations naturally being reticent to publish details relating to failures with their products. However within a single organisation it should be possible to capture and share this knowledge for the wider benefit without compromising the competitive position, indeed this should reduce risk from loss of important knowledge and experience when individuals leave the organisation.

A limitation of the case study is that as the base data has been given in confidence it was not possible to peer review and cross check the categorisation decisions. Future work could include correlation studies using the analysis method and categories developed on new data sources by more than one person to confirm the repeatability and reproducibility of the results.

One potential method for capturing this knowledge is through robustness cases with robustness objectives to mitigate known issues and solutions representing design patterns

which have been found to be effective in achieving this. It was identified that the argument modules have potential for re-use but study of further applications of these could investigate the levels that can be actually achieved and the factors that promote or inhibit re-use. Additionally there may be other mechanisms for systematically capturing and codify domain knowledge which is a potential area for further work.

8.3.2

8.3.3 Applicability of methods to other domains

The methods developed in this thesis have been targeted at automotive electronic systems but in principle could be applicable to other domains with similar issues. Potential areas could be information technology, consumer goods, and non-safety critical applications in aerospace such as in-flight infotainment systems.

8.3.4 Integration of Robustness Models with Other Modelling

As noted in the literature review SysML is increasingly being used for modelling at systems level but lacks detail at a behavioural level which would allow emergent behaviour to be detected. At a component level Matlab Simulink is increasingly being used to model detailed functionality from which application software can be derived through auto coding. By combining the architectural and systems interaction information from SysML model with the detailed behavioural information on existing functions from component level Matlab Simulink models there may be an opportunity to more rapidly develop robustness models.

8.3.5 Application of formal model checkers to robustness models

It was noted that the use of formal methods have significant potential within automotive and are the subject of on-going pilot and research projects focusing initially on high integrity applications. This should pave the way for lightweight methods and tools

which then can be used for robustness analysis of all systems. A potential area of future work is to conduct formal analysis using model checkers on robustness models which encompass robustness critical attributes and the knowledge of what are the specific properties that need to be checked to prove robustness.

8.3.6 Application of Robustness Modelling Techniques to Functional Safety Analysis

Converse to the use of techniques from safety analysis domain, such as safety cases, for design for robustness there also may be an opportunity to use robustness modelling techniques for complex systems for functional safety analysis. Through the work on robustness modelling a framework was developed for modelling and verifying distributed system functionality whereby a function is delivered through the interaction of several ECUs across one or more communication buses. This modelling framework could be used for modelling the behaviour of safety critical functions across a distributed system which could be for checking completeness of functional specifications, early detection of emergent behaviour and as evidence to support a safety case analogous to the interaction between the robustness case and robustness models.

8.3.7 Abuse case elicitation techniques

An issue identified in the case studies was that of unforeseen use cases where the user deviated from intended operation of the system. This is an inherent risk where complex systems are used by untrained users. A potential area for further work is in the development of a structured methodology for elicitation techniques for unintended use cases of automotive systems.

REFERENCES

- Afzal, W., Torkar, R., and Feldt, R. (2009). *A systematic review of search-based testing for non-functional system properties*. Information and Software Technology, 51(6), 957-976.
- Amor-Segan, M. L., McMurran, R., Dhadyalla, G., and Jones, R. P. (2007). *Towards the self healing vehicle*. In Automotive Electronics, 2007 3rd Institution of Engineering and Technology Conference on (pp. 1-7). IET.
- Andrews Z, Ingram C, Payne R, Romanovsky A, Holt J and Perry S. (2014). *Traceable Engineering of Fault-Tolerant SoSs*. In: INCOSE International Symposium 2014. 2014, Nevada, USA: International Council on Systems Engineering (INCOSE).
- Arshad, N., Heimbigner, D., and Wolf, A. L. (2004). *A planning based approach to failure recovery in distributed systems*. In Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems (pp. 8-12). ACM.
- Artop (2013). *AUTOSAR tool platform*. Retrieved May 29, 2013, from <http://www.artop.org>
- Arora, A., and Gouda, M. (1994). *Distributed reset*. Computers, IEEE Transactions on, 43(9), 1026-1038.
- Avižienis, A., Laprie, J. C., Randell, B., & Landwehr, C. (2004). *Basic concepts and taxonomy of dependable and secure computing*. Dependable and Secure Computing, IEEE Transactions on, 1(1), 11-33.
- ATESST2 (2010) *ATESST2 project brochure*. Available from ATESST project website. Retrieved May 9, 2013, from <http://www.atesst.org>

AUTOSAR (2013), *AUTOSAR*. Retrieved May 29, 2013, from <http://www.autosar.org>

Barker, S, Kendall, I. and Darlison. A. (1997). *Safety cases for software-intensive systems: an industrial experience report*. In Safe Comp 97 (pp. 332-342). Springer London.

Bar-Yam, Y., Allison, M., Batdorf, R., Chen, H., Generazio, H. Singh, H., Tucker, S. (2004). *The characteristics and emerging behaviors of system of systems*. NECSI: Complex Physical, Biological and Social Systems Project, Proceedings of the 2006 IEEE International Conference on System of Systems Engineering, Los Angeles, CA, USA - April 2006.

Bates, S., Bate, I. J., Hawkins, R. D., Kelly, T. P., and McDermid, J. A. (2003). *Safety case architectures to complement a contract-based approach to designing safe systems*. In 21st International System Safety Conference, System Safety Society.

Becci, G., Dhadyalla, G., Mouzakitis, A., Marco, J., and Moore, A. D. (2013). *Robustness testing of real-time automotive systems using sequence covering arrays*. SAE International Journal of Passenger Cars-Electronic and Electrical Systems, 6(1), 287-293.

Blochwitz, T., Otter, M., Akesson, J., Arnold, M., Clauß, C., Elmqvist, H., Freidrich, M., Junghanns, A., Mauss, J., Neumerkel, D., Olsson, H., and Viel, A. (2012). *Functional Mockup Interface 2.0: The standard for tool independent exchange of simulation models*. In 9th International Modelica Conference, Munich.

Boardman, J. (2006). *System of Systems – the meaning of of*. Proceedings of the 2006 IEEE International Conference on System of Systems Engineering, Los Angeles, CA, USA - April 2006.

- Box, G.E.P. (2000). *CQPI Report No 179 Statistics for discovery*. Center for Quality and Productivity Improvement (CQPI), University of Wisconsin, Madison, Wisconsin. Retrieved April 7, 2013, from <http://www.engr.wisc.edu/centers/cqpi/>
- Bringmann, E., and Krämer, A. (2006, May). *Systematic testing of the continuous behavior of automotive systems*. In Proceedings of the 2006 international workshop on Software engineering for automotive systems (pp. 13-20). ACM.
- Broenink, J. F., Kleijn, C., Larsen, P. G., Jovanovic, D., Verhoef, M., and Pierce, K. (2010). *Design support and tooling for dependable embedded control software*. In Proceedings of the 2nd International Workshop on Software Engineering for Resilient Systems (pp. 77-82). ACM.
- Brown, S. (2000). *Overview of IEC 61508. Design of electrical/electronic/programmable electronic safety-related systems*. Computing and Control Engineering Journal, 11(1), 6-12.
- Bryans J, Fitzgerald J, Payne R, and Kristensen K. (2014). *Maintaining Emergence in Systems of Systems Integration: a Contractual Approach using SysML*. In: INCOSE International Symposium (IS2014). 2014, Las Vegas, Nevada.
- Butler, R., Finelli, G. (1991). *The infeasibility of experimental quantification of life-critical software reliability*. ACM SIGSOFT Software Engineering Notes, Vol. 16, pp.66–76.
- Charette, R. N. (2009). *This car runs on code*. IEE Spectrum, Feb 2009.
- Chen, D., Feng, L., Qureshi, T. N., Lönn, H., and Hagl, F. (2013). *An architectural approach to the analysis, verification and validation of software intensive embedded systems*. Computing, 1-40. Springer-Verlag.

Coleman, J. W., Malmos, A. K., Larsen, P. G., Peleskay, J., Hains, R., Andrews, Z., ... and

Didier, A. (2012). *COMPASS tool vision for a system of systems collaborative development environment*. In System of Systems Engineering (SoSE), 2012 7th International Conference on (pp. 451-456). IEEE.

Cuenot, P., Gerard, S., Lonn, H., Reiser, M. O., Servat, D., Sjostedt, C. J., Kolagari, R.T.,

Törngren, M. and Weber, M. (2007). *Managing complexity of automotive electronics using the East-ADL*. In Engineering Complex Computer Systems, 2007. 12th IEEE International Conference on (pp. 353-358). IEEE.

Despotou, G. (2011). *Introducing the challenges of dependable systems of systems*.

Workshop on Dependable Systems of Systems, University of York, 5-6 September 2011.

Dhadyalla, G., McMurrin, R., Amor-Segan, M., Li, W., Talbot, K., Jones, R.P. (2010).

Robustness testing against low voltage transients - a novel approach. SAE Technical Paper 2010-01-0195, 2010, doi: 10.4271/2010-01-0195.

Dhadyalla, G. (2013). “*Robustness testing of real-time automotive systems using sequence covering arrays*”. SAE 2013 World Congress and Exhibition, Paper 13AE-0121, April, 2013, Detroit, Michigan, USA.

Despotou, G., and Kelly, T. (2004). *Extending the safety case concept to address*

dependability. In Proceedings of the 22nd International System Safety Conference (pp. 645-654). System Safety Society Publications, P.O. Box 70, Unionville, VA 22567-0070.

Driscoll, K., Hall, B., Sivencrona, H., and Zumsteg, P. (2003). *Byzantine fault tolerance, from theory to reality*. In Computer Safety, Reliability, and Security (pp. 235-248).

Springer Berlin Heidelberg.

- Fernandez, J. C., Mounier, L., and Pachon, C. (2005). *A model-based approach for robustness testing*. In *Testing of Communicating Systems* (pp. 333-348). Springer Berlin Heidelberg.
- Fitzgerald, J. S., Larsen, P. G., Pierce, K. G., and Verhoef, M. H. G. (2013). *A formal approach to collaborative modelling and co-simulation for embedded systems*. *Mathematical Structures in Computer Science*, 23(04), 726-750.
- Fitzgerald, J., Larsen, P. G., and Woodcock, J. (2014). *Foundations for Model-Based Engineering of Systems of Systems*. In *Complex Systems Design & Management* (pp. 1-19). Springer International Publishing.
- Frischkorn, H. (2004). *Automotive software the silent revolution*. Automotive Software Workshop, San Diego, 2004. Retrieved May 28, 2013, from <http://aswsd.ucsd.edu/2004/program.html>
- Fritzsche, R. (2006). *Using parameter-diagrams in automotive engineering*. *ATZ worldwide*, 108, no.6 (2006), 17-21.
- GENIVI (2013), *GENIVI website*. Retrieved May 29, 2013, from <http://www.genivi.org>
- Grochtmann, M., Grimm, K. and Wegener, J. (1993). *Tool-supported test case design for black-box testing by means of the classification-tree editor*. *Proceedings of EuroSTAR*, 93, 169-176.
- Grzemba, A, ed. (2008). *MOST - the automotive multimedia network..*, Editor Prof. Dr. Ing. Andreas Grzemba, 2008 Franzis Verlag Gmbh, 85586 Poing, ISBN 973-3-7723-5316-1, Retrieved Jan 20, 2009, from <http://www.mostcooperation.com>

GSN Working Group (2011). *GSN Community Standard Version 1 November 2011*.

Retrieved May 30, 2014, from <http://www.goalstructuringnotation.info>

GSN Working Group (2012). *GSN Working Group Online - resources tools*. Retrieved Feb

2, 2012, from <http://www.goalstructuringnotation.info>

Habli, I., Hawkins, R., and Kelly, T. (2010). *Software safety: relating software assurance and software integrity*. International Journal of Critical Computer-Based Systems, 1(4), 364-383.

Habli, I., Ibarra, I., Rivett, R., and Kelly, T. (2010). *Model-based assurance for justifying automotive functional safety*. In Proc. 2010 SAE World Congress.

Hank, P., Suermann, T., Müller, S. (2012) *Automotive Ethernet, a Holistic Approach for a Next Generation In-Vehicle Networking Standard*, Advanced Microsystems for Automotive Applications 2012, pp 79-89

Hayhurst, K., Veerhusen, D., Chilenski, J., Rierson, L.(2001). *A practical tutorial on Modified Condition/ Decision Coverage*. NASA Scientific and Technical Information (STI) Program Office Report NASA/TM-2001-210876, May 2001.

Heinecke, H., Damm, W., Josko, B.; Metzner, A., Kopetz, H., Sangiovanni-Vincentelli, A., Di Natale, M. (2008). *Software components for reliable automotive systems design, automation and test in Europe*. DATE '08 , vol., no., pp.549,554, 10-14 March 2008.

Highsmith, J. (2004). *Agile project management*. Addison-Wesley ISBN 0-321-21977-5, 2004.

- Hyung-Taek L, Volker, L., Herrscher, D. (2011). *Challenges in a future IP/Ethernet-based in-car network for real-time applications*. Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE , vol., no., pp.7,12, 5-9 June 2011.
- Huang, Y., McMurran, R., Dhadyalla, G., Jones, R.P. (2009). *Automated functional and robustness testing of vehicle infotainment systems*. SAE 2009-01-1366, April 2009.
- Huang, Y., McMurran, R., Amor-Segan, M., Dhadyalla, G., Jones, R.P., Bennett, P., Mouzakitis, A., and Kieloch, J. (2010). *Development of an automated testing system for vehicle infotainment system*. The International Journal of Advanced Manufacturing Technology Volume 1 / 1985 - Volume 51 / 2010.
- Forder, R. (2012). *What is model based system engineering*. INCOSE UK Leaflet Z9 Issue 1.0 January 2012, Retrieved April 27, 2013, from <http://www.incoseonline.org.uk>
- International Electrotechnical Commission (IEC) (2005). *IEC/TR 61508 – Functional safety of electrical/electronic/programmable electronic safety-related systems*. Published in UK by the British Standards Institute, ISBN 978 0 580 50917 9.
- International Standards Organisation (ISO) (2011). *ISO26262 - Road vehicles — functional safet.*, Published in UK by the British Standards Institute, ISBN 978 0 580 62303 5.
- Jackson, D., Thomas, M., and Millett, L. I. (Eds.) (2007). *Software for Dependable Systems: Sufficient Evidence?* National Academies Press.
- Jaguar-Land-Rover (2013). *Jaguar Land-Rover Careers*.
<http://www.jaguarlandrovercareers.com> - accessed 29/9/13

J.D.Power (2013). *2013 Vehicle Dependability Survey Press release*.

www.jdpower.com/dependability - accessed 18/10/13.

Johannesson, P., Bergman, B., Svensson, T., Arvidsson, M., Lönnqvist, Å., Barone, S., and Maré, J. (2013). *A Robustness Approach to Reliability*. Quality and Reliability Engineering International, 29(1), 17-32.

Kelly, T. P. (2001). *Concepts and principles of compositional safety case construction*. Contract Research Report for QinetiQ COMSA/2001/1/1.

Kelly, T. P., and McDermid, J. A. (1997). *Safety case construction and reuse using patterns*. In Safe Comp 97 (pp. 55-69). Springer London.

Kelly, T., and Weaver, R. (2004). *The goal structuring notation—a safety argument notation*. In Proceedings of the Dependable Systems and Networks 2004 Workshop on Assurance Cases.

Knight, J., Heimbigner, D., Wolf, A. L., Carzaniga, A., Hill, J., Devanbu, P., and Gertz, M. (2001). *The Willow architecture: comprehensive survivability for large-scale distributed applications*. Colorado Univ at Boulder, Dept of Computer Science.

Knüchel, C., Rudorfer, M., Voget, S., Eberle, S., Sezestre, R., Loyer, A. (2010). *Artop – An ecosystem approach for collaborative AUTOSAR tool development*. ERTS 2010, Toulouse, May 2010.

Kopetz, H. (1999). *Elementary versus composite interfaces in distributed real-time systems*. In The Fourth International Symposium on Autonomous Decentralized Systems, IEEE Computer Society, Tokyo, Japan, March 1999. 1,14,15.

- Kopetz, H. (2011). *Real-time systems - design principles for distributed embedded applications*. Second Edition, Springer, ISBN 978-1-4419-8236-0, 2011.
- Lamport, L., Shostak, R., and Pease, M. (1982). *The Byzantine generals problem*. ACM Transactions on Programming Languages and Systems (TOPLAS), 4(3), 382-401.
- Larman, C. (2004). *Agile and iterative development*. Addison-Wesley ISBN 0-13-111155-8, 2004.
- Lehmann, E., and Wegener, J. (2000). *Test case design by means of the CTE XL*. In Proceedings of the 8th European International Conference on Software Testing, Analysis and Review (EuroSTAR 2000), Copenhagen, Denmark.
- Lehmann, E. (2000). *Time partition testing: A method for testing dynamic functional behaviour*. Proceedings of TEST2000, London, Great Britain.
- Leyuan, S., Chun-Hung, C., Enver, Y. (1999). *Simultaneous simulation experiments and nested partition for discrete resource allocation in supply chain management*. Proceedings of the 31st conference on Winter simulation: Phoenix, Arizona, United States, Volume 1 Pages: 395 – 401, 1999, ISBN:0-7803-5780-9.
- Li, Z., Sim, C. H., and Low, M. Y. H. (2006). *A survey of emergent behavior and its impacts in agent-based systems*. In Industrial Informatics, 2006 IEEE International Conference on (pp. 1295-1300). IEEE.
- Lisagor, O., Pretzer, M., Seguin, C., Pumfrey, D. J., Iwu, F. and Peikenkamp, T. (2006). *towards safety analysis of highly integrated technologically heterogeneous systems – a domain-based approach for modelling system failure logic*. In 24th International System Safety Conference (ISSC), August 2006.

- Lissack, M., Roos, J. (2000). *The next common sense: the e-manager's guide to mastering complexity*. Nicholas Brealey Publishing, London, 2000.
- Littlewood, B., Strigini, L. (1993). *Validation of ultrahigh dependability for software-based systems*, Communications of the ACM, Vol. 36, No. 11, pp.69–80.
- Lucas, Chris. (2000). *Quantifying complexity theory*. CALResCo Group. Retrieved April 7, 2013, from <http://www.calresco.org/lucas/quantify.htm>
- Maier, M.W. (1998). *Architecting principles for systems of systems*. Systems Engineering, Vol. 1, No. 4, 1998, pp. 267-284.
- Marstaller, R. (2013). *Potentials of function network analysis on vehicle level*. 3rd International Conference Applying ISO 26262, Munich, March 2003.
- Matsubara, M., Sakurai, M., Narisawa, F., Enshoiwa, M., Yamane, Y. and Yamanaka, H. (2013). *Application of model checking to automotive control software with slicing technique*. SAE 2013 World Congress, paper 2013-01-0436, Published 04/08/2013.
- McConnell, S. (2004). *Code complete*. 2nd Edition, Microsoft Press, 2004.
- McMurrin, R., Jones, R.P. (2010) “*Robustness Modelling of Automotive Electrical Systems of Systems*”, FISITA 2010, 30th May - 4th June 2010, Budapest, Hungary.
- McMurrin, R., Leslie, J. (2010) “*MOST Control Channel Modelling*”, MOST Forum 2010, 23rd March 2010, Frankfurt Germany, Reprinted in Elektronik automotive March 2010 Special issue MOST.

- McMurrin, R., Jones, R. (2013) “*Robustness Modelling of Complex Systems - Application to the Initialisation of a Hybrid Electric Vehicle Propulsion System*” SAE Technical Paper 2013-01-1231, 2013, doi:10.4271/2013-01-1231.
- McMurrin, R. McKinney, F. Tudor, N. Milam, W. P. (2006). *Dependable systems of systems*. SAE Transactions 2007, Vol 115; Part 7, pages 287-294, ISBN ISSN 0096-736X.
- Meyer, B. (1997). *Object oriented software construction*. 2nd Edition, Prentice-Hall, 1997.
- Mössinger, J. (2010). *Software in automotive systems*. IEEE Software, Volume:27 , Issue: 2, ISSN 0740-7459, 2010.
- MOST Cooperation (2005). *MOST Specification Rev 2.4*. MOST Cooperation, Karlsruhe, Germany, May 2005. Retrieved Jan 20, 2009, from the MOST Co-operation website <http://www.mostcooperation.com>
- NASA (2011). *National Highway Traffic Safety Administration Toyota unintended acceleration investigation*. NASA Engineering and Safety Center Technical Assessment Report, NESC Assessment #: TI-10-00618.
- Oasis (2003). *OS8104 MOST network transceiver final product data sheet*. Oasis Silicon Systems, Jan 2003.
- Oliveira M., Sampaio A., Antonino P., Ramos R., Cavalcanti A. and Woodcock J. (2012). *Compositional analysis and design of CML Models*. COMPASS Deliverable Number: D24.1, Version: 0.4, March, 2012, Retrieved March 21, 2013, from <http://www.compass-research.eu>

- Palin, R., and Habli, I. (2010). *Assurance of automotive safety—a safety case approach*. In Computer Safety, Reliability, and Security (pp. 82-96). Springer Berlin Heidelberg.
- Pande, Peter S., Neuman, Robert P., Roland R. Cavanagh (2001). *The Six Sigma way: how GE, Motorola, and other top companies are honing their performance*. New York, NY: McGraw-Hill Professional. ISBN 0071358064.
- Ponticel, P. (2013). *Global viewpoints – North America brands are king*. Interview with Mark Chernoby – Senior VP Engineering Chrysler, Automotive Engineering International, April 2, 2013, SAE International.
- Powell, D. (2010). *A generic fault-tolerant architecture for real-time dependable systems*. Springer Publishing Company, Incorporated.
- Pretschner, A., Broy, M., Krüger, I.H., Stauner, T. (2007). *Software engineering for automotive systems: a roadmap*. FOSE '07 2007 Future of Software Engineering.
- Rajeev, A. C., Mohalik, S., Ramesh, S., (2013). *Design verification of automotive controller models*. SAE Int. J. Passeng. Cars – Electron. Electr. Syst. 6(2):2013, doi:10.4271/2013-01-0428.
- Rao, A. C., McMurran, R., and Jones, R. P. (2009). *A critical analysis of model-based formal verification efforts within the automotive industry*. SAE International Journal of Passenger Cars-Electronic and Electrical Systems, 1(1), 77-83.
- Rau, M., Weiß, P. (2008) *Development of an automotive standard with focus on functional safety (ISO 26262)*. Proceedings FISITA 2008.

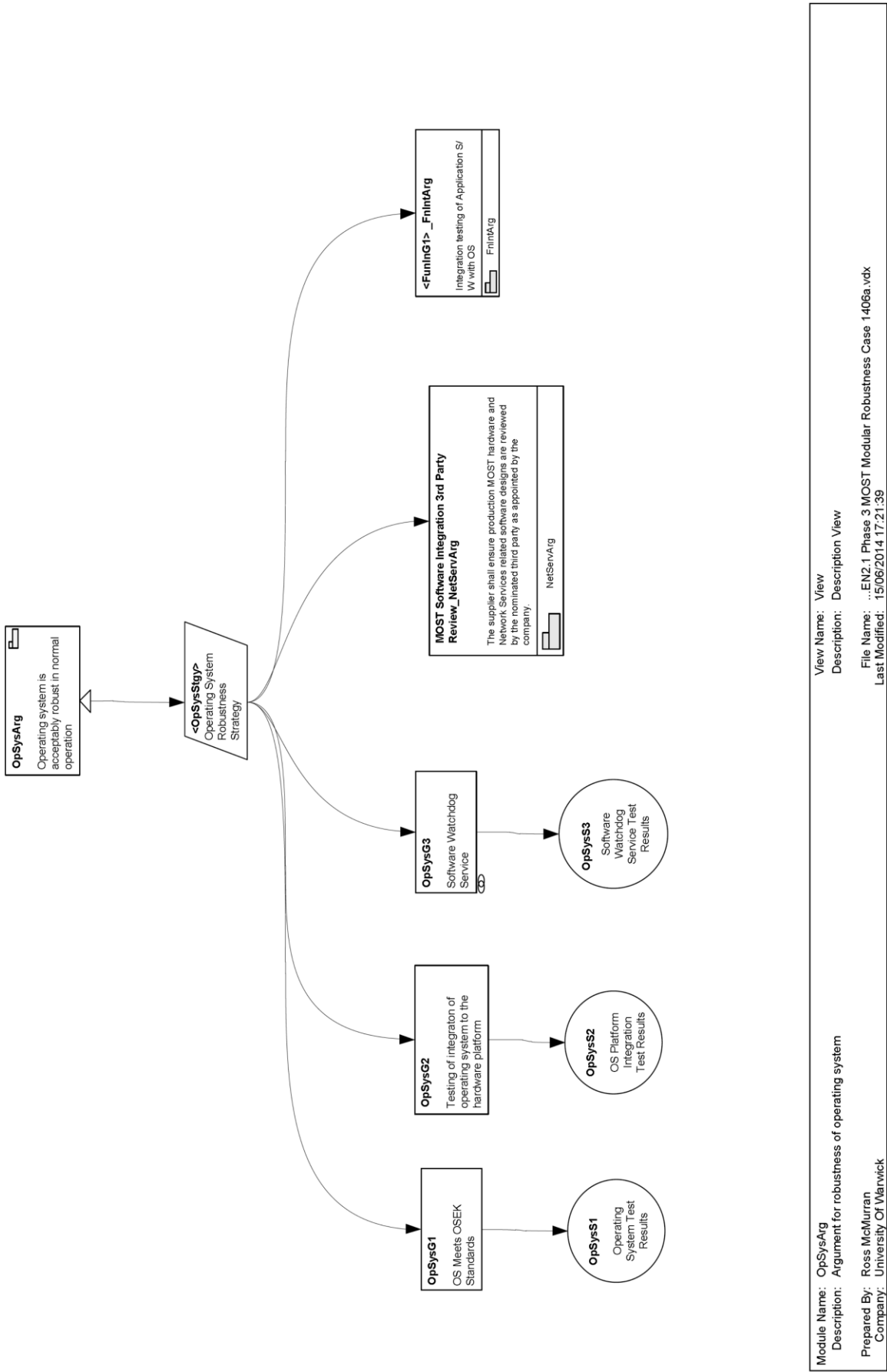
- Reichart, G. Heinecke, H. (2006). *Systemintegration – im wechselspiel von architektur technologie und prozess*. 10. Jahrestagung Euroforum 07.02.2006.
- Reuters (2013). *GM recalls about 33,700 vehicles to fix software issue*. Reuters. Retrieved May 30, 2013, from <http://www.reuters.com/article/2013/03/20/us-gm-recall-idUSBRE92J0HC20130320>
- Ricardo (2013). *PICASSOS project press release*. Retrieved November 20, 2013, from <http://www.ricardo.com/en-GB/News--Media/Press-releases/News-releases1/2013>
- Rivett, R. (2013). *The OEM/supplier relationship: An OEM perspective*. 3rd International Conference Applying ISO 26262, Munich, March 2003.
- Rushby, J. (2002). *An overview of formal verification for the time-triggered architecture*. Invited paper presented at FTRTFT'02, Oldenburg, Germany, September 2002. Springer-Verlag LNCS Vol. 2469, pp. 83–105. Springer-Verlag.
- SAE Automotive Quality and Process Improvement Committee (2009). *J1739 Potential Failure Mode and Effects Analysis in Design (Design FMEA), Potential Failure Mode and Effects Analysis in Manufacturing and Assembly Processes (Process FMEA)*. SAE International.
- Schedl, A. (2007). *Goals and architecture of Flexray at BMW*. In slides presented at the Vector FlexRay Symposium 2007.
- Scheickl, O., Ainhauser, C. and Gliwa, P. (2012). *Tool support for seamless system development based on AUTOSAR timing extensions*. ERTS 2012 - Embedded Real Time Software and Systems, Toulouse, February 2012.

- Schulz, A. P., and Fricke, E. (1999). *Incorporating flexibility, agility, robustness, and adaptability within the design of integrated systems-key to success?* In Digital Avionics Systems Conference, 1999. Proceedings. 18th (Vol. 1, pp. 1-A). IEEE.
- Stolle, R., Salzmann, C. and Tillmann S. (2006). *Mastering complexity through modelling and early prototyping*. 7. Euroforum-Jahrestagung Software im Automobile, May 2006.
- Taylor, J.E., Amor-Segan, M., Dhadyalla, G. and Jones, R.P. (2012) *Discerning the operational state of a vehicle's distributed electronic systems from vehicle network traffic for use as a fault detection and diagnosis tool*. AVEC 2012, 9-12 Sept 2012, Seoul, South Korea.
- Thompson, W, (Lord Kelvin) (1904). *Baltimore lectures on molecular dynamics and the wave theory of light*. 1904.
- Toyota (2011). *Toyota Annual Report 2011*. Toyota Motor Company. Retrieved November 19, 2013, from http://www.toyota-global.com/investors/financial_result/2011/
- Wang, W., Wu, J. Y-T. and Lust, R. V. (1997). *Deterministic design reliability-based design and robust design*. MSC 1997 Aerospace Users' Conference Proceedings. Retrieved May 30, 2013, from www.mssoftware.com
- Warranty Direct Reliability Index (2013). *Manufacturer ratings – electrical*. Retrieved May 29, 2013, <http://www.reliabilityindex.com/manufacturer/Electrical>
- Weaver, W. (1948). *Science and complexity*. In American Scientist, 36: 536 (1948).
- Woodcock, J., Larsen, P. G., Bicarregui, J., & Fitzgerald, J. (2009). *Formal methods: Practice and experience*. ACM Computing Surveys (CSUR), 41(4), 19.

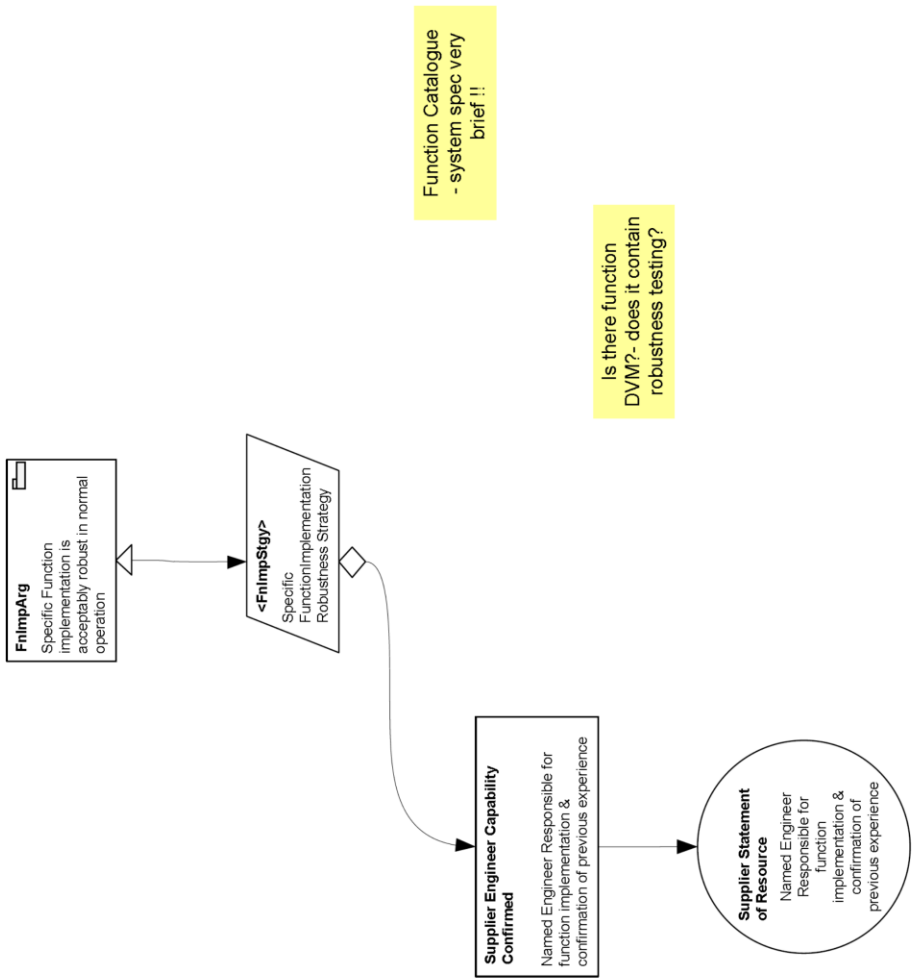
Yang, K. and El-Haik, B. (2003). *Design for Six Sigma - a roadmap for product development*.
New York: McGraw-Hill.

Infotainment Robustness Case - Top Level Argument

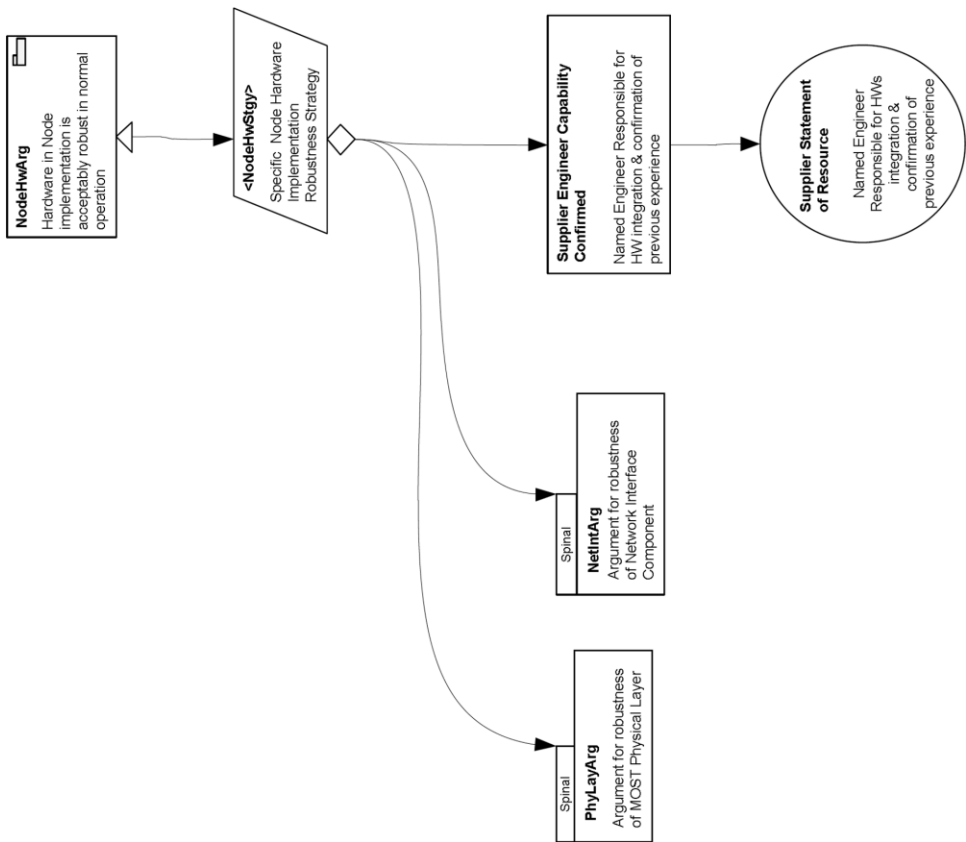




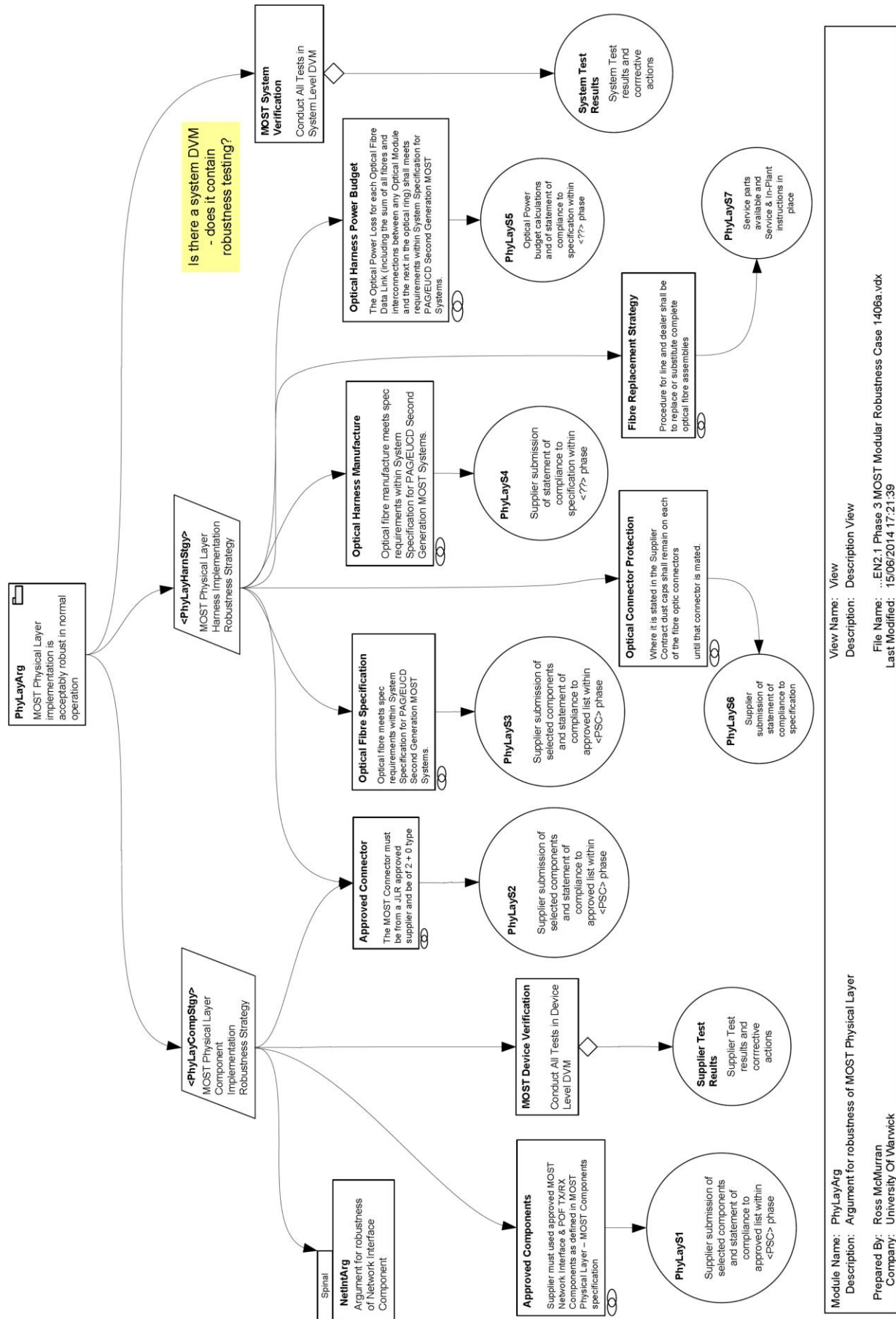
Infotainment Robustness Case – Function Implementation Argument



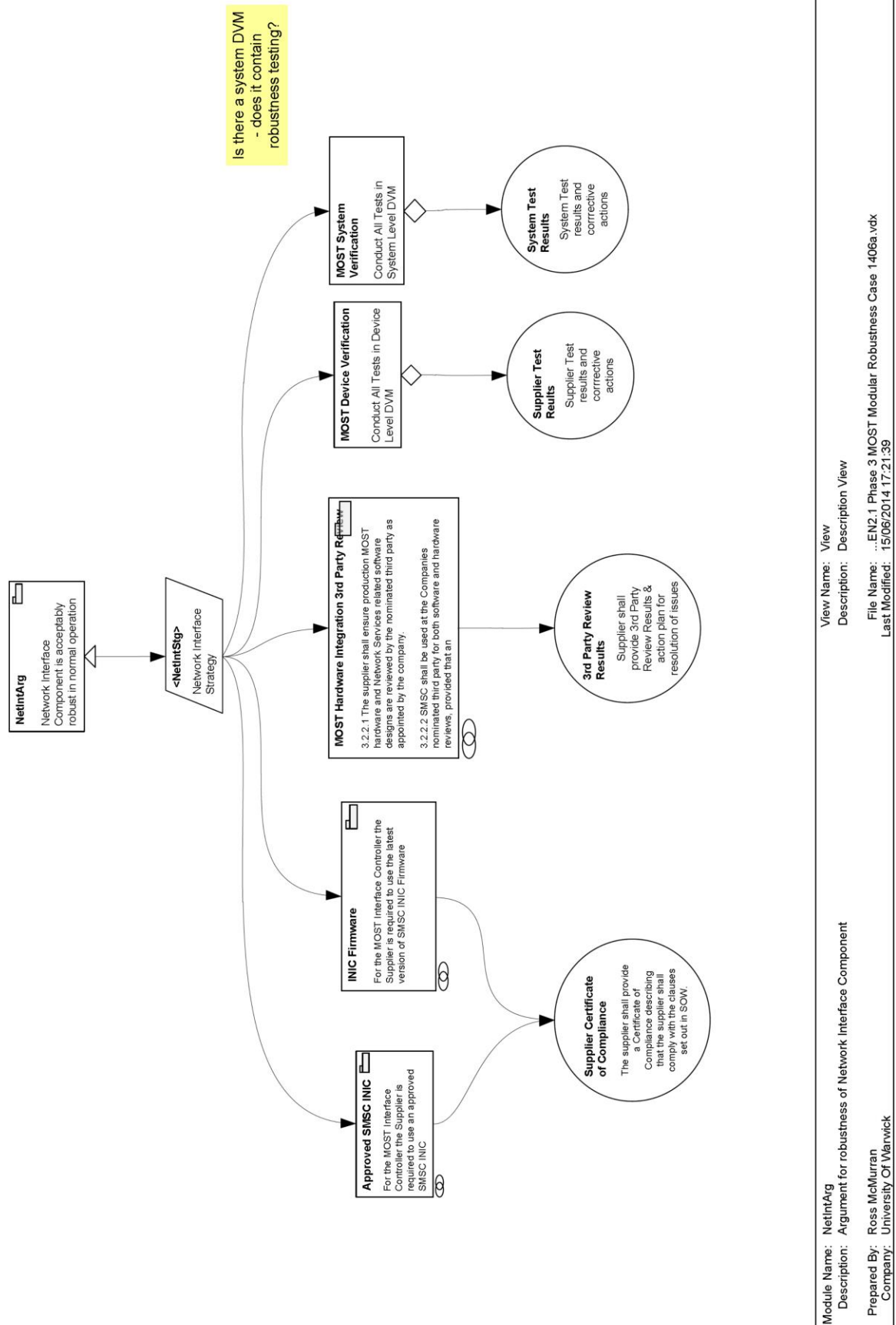
Module Name: FImpArg	View Name: View
Description: Argument for robustness of implementation of specific functions	Description: Description View
Prepared By: Ross McMullan	File Name: ..EN2.1 Phase 3 MOST Modular Robustness Case 1406a.vdx
Company: University Of Warwick	Last Modified: 15/06/2014 17:21:39



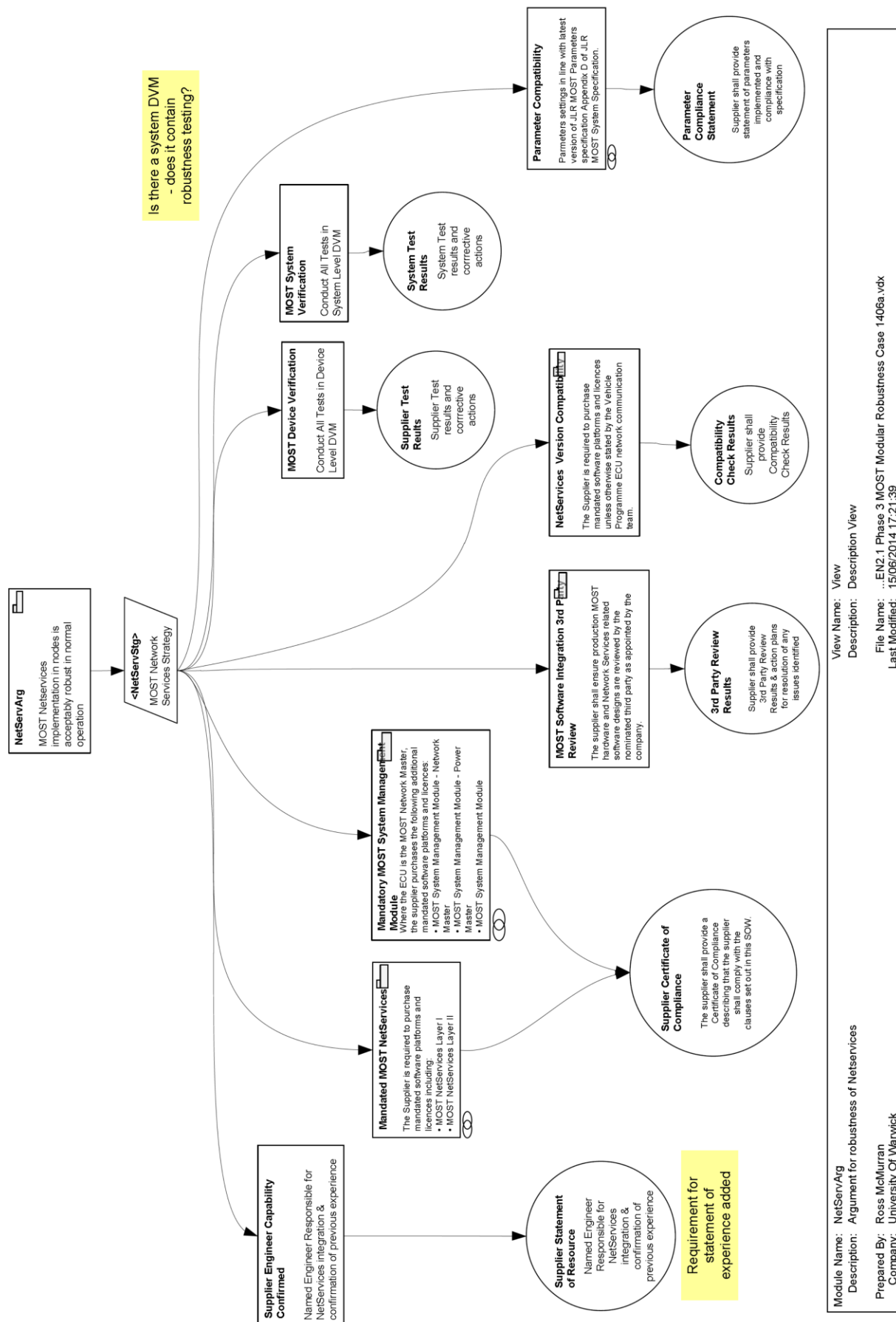
Infotainment Robustness Case – Argument for Robustness of MOST Physical Layer



Infotainment Robustness Case – Argument for Robustness of Network Interface Controller

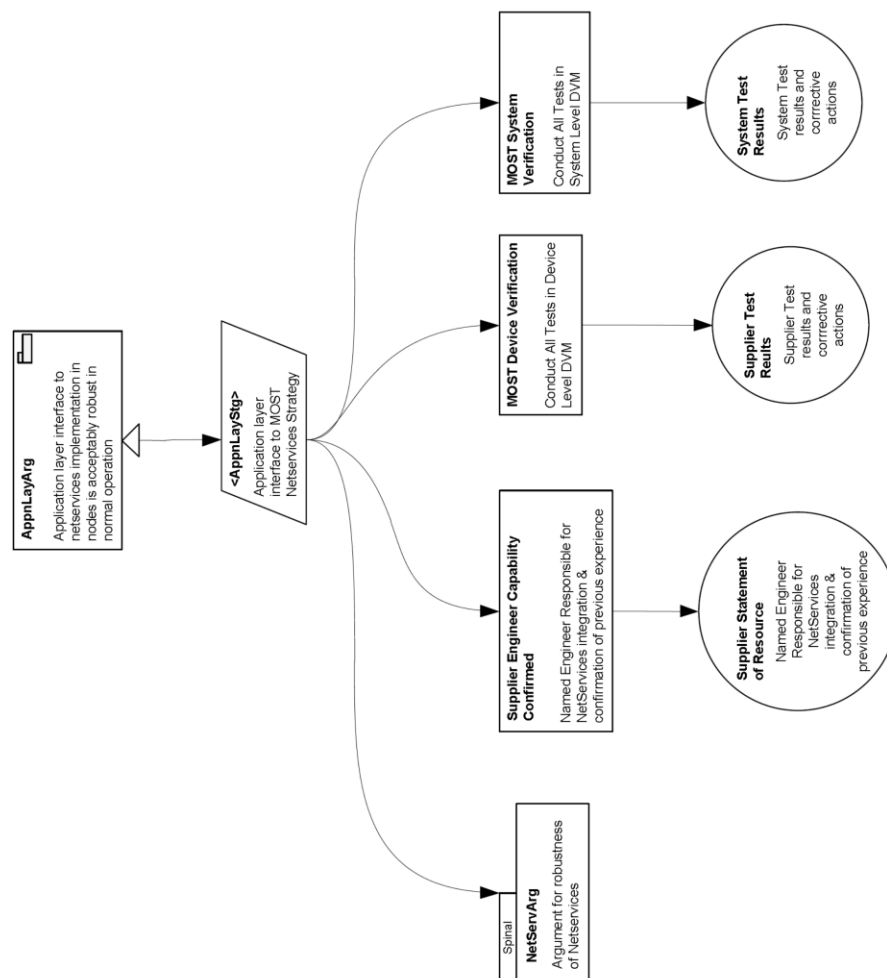


Infotainment Robustness Case – Argument for Robustness of Net Services

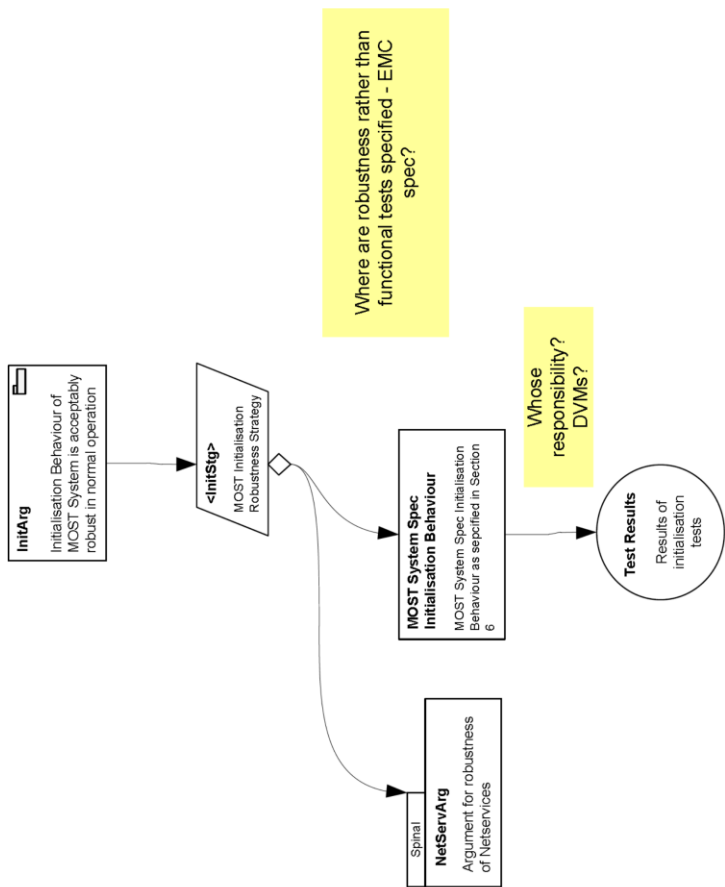


Infotainment Robustness Case – Argument for Robustness of Integration of Application

Layer Interface to Net Services

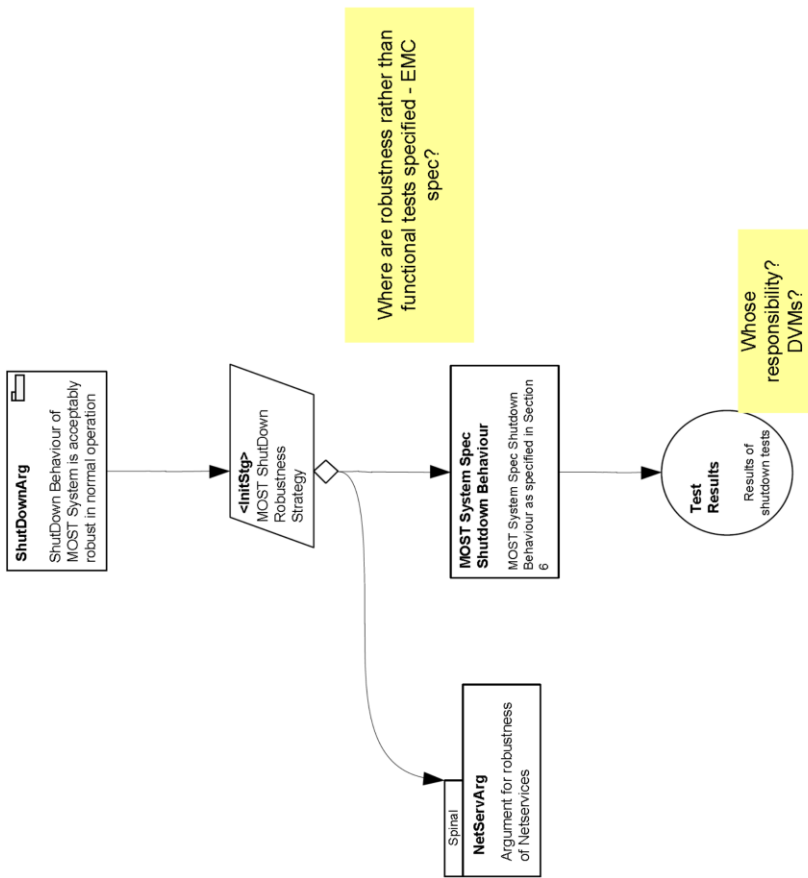


Module Name: AppnLayerArg	View Name: View
Description: Argument for robustness of application layer interface to net services	Description: Description View
Prepared By: Ross McMullan	File Name: ..EN2.1 Phase 3 MOST Modular Robustness Case 1406a.vdx
Company: University Of Warwick	Last Modified: 15/06/2014 17:21:39



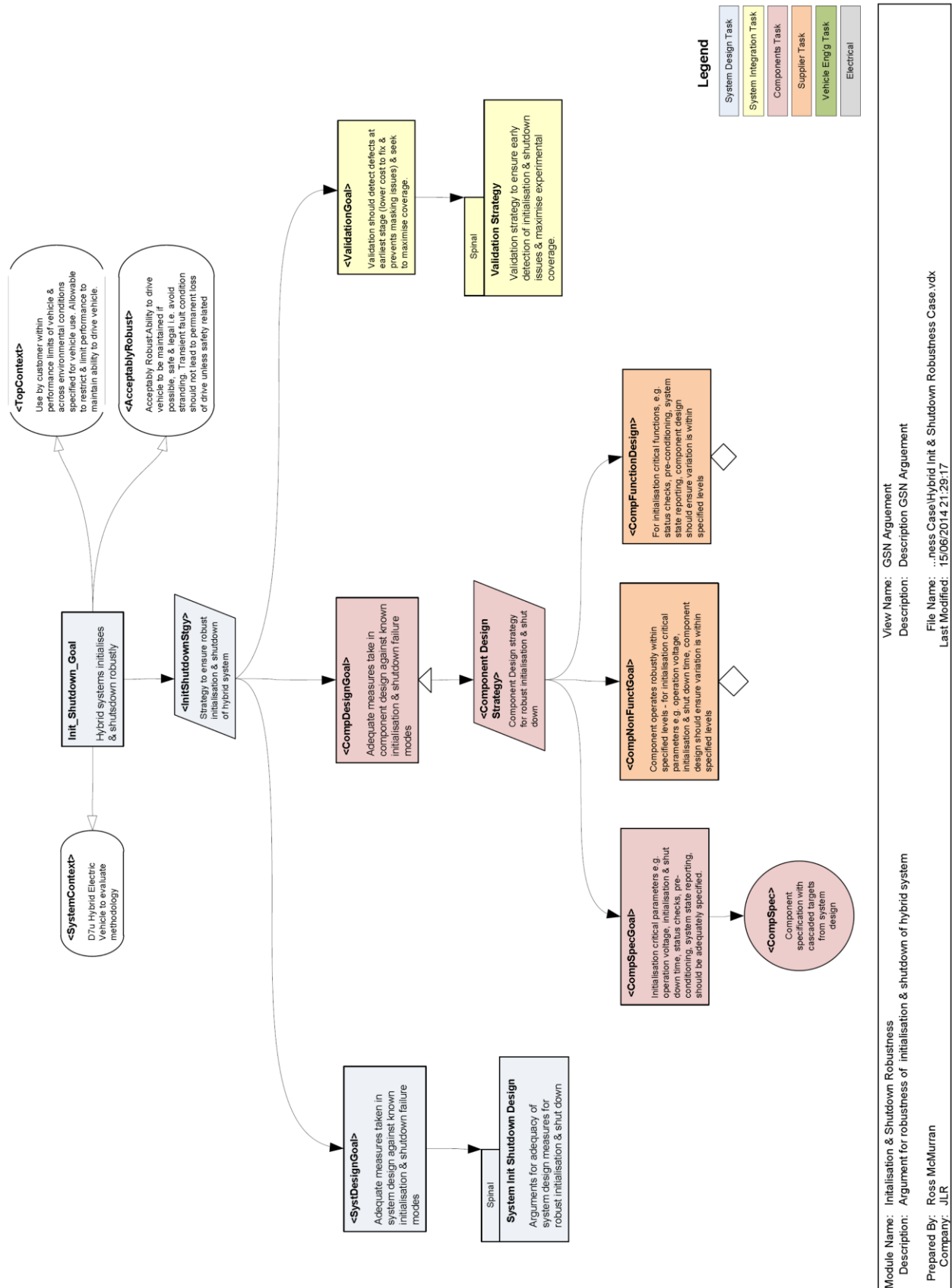
Module Name: InitArg
Description: Argument for robustness of initialisation behaviour
Prepared By: Ross McMullan
Company: University Of Warwick

View Name: View
Description: Description View
File Name: ..EN2.1 Phase 3 MOST Modular Robustness Case 1406a.vdx
Last Modified: 15/06/2014 17:21:39

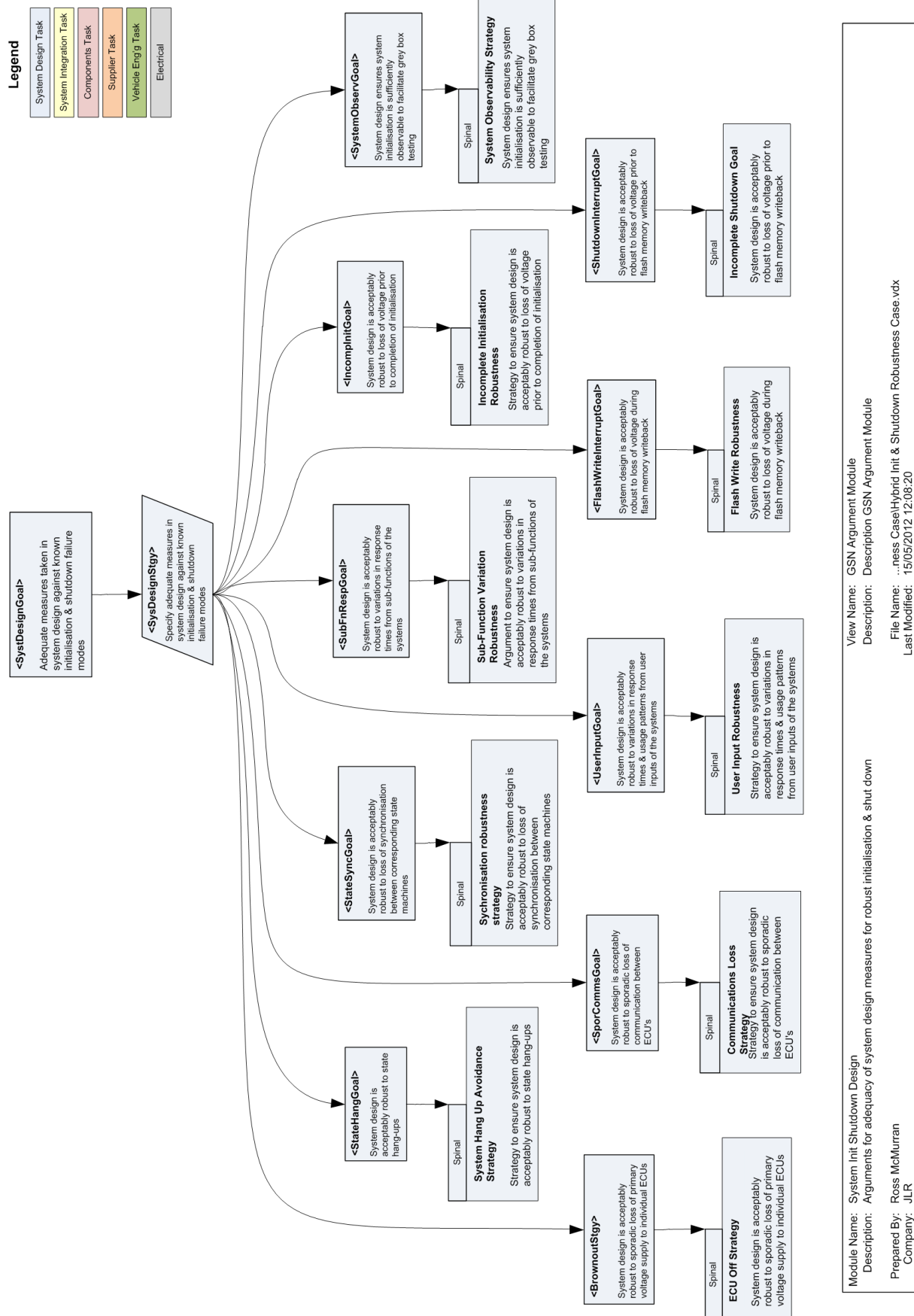


Appendix 2 – Hybrid System Initialisation Robustness Case

Hybrid System Initialisation Robustness Case – Top Level Argument

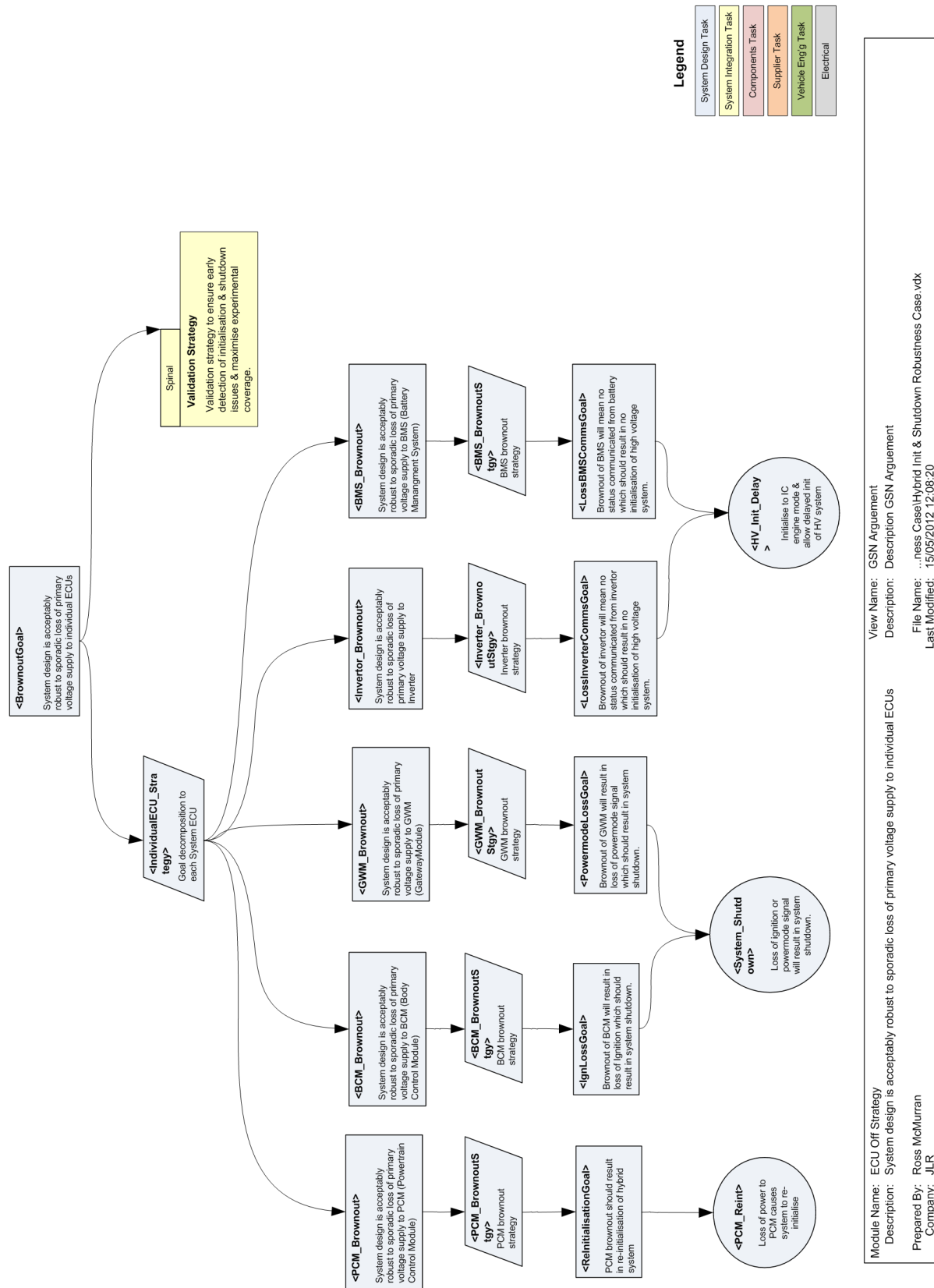


Hybrid System Initialisation Robustness Case – Overall System Design Argument

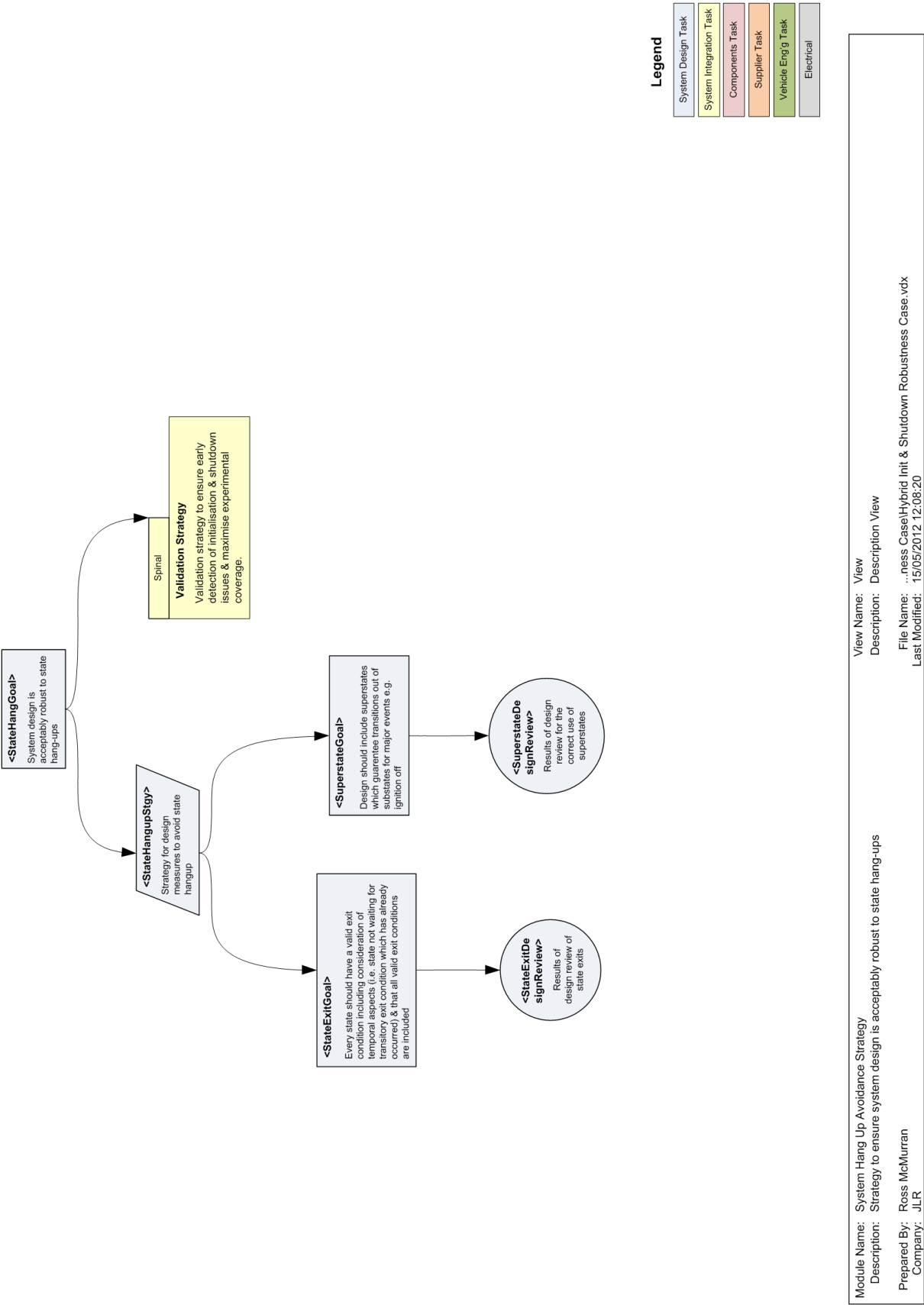


Hybrid System Initialisation Robustness Case – Argument for Robustness to Sporadic

Voltage Loss

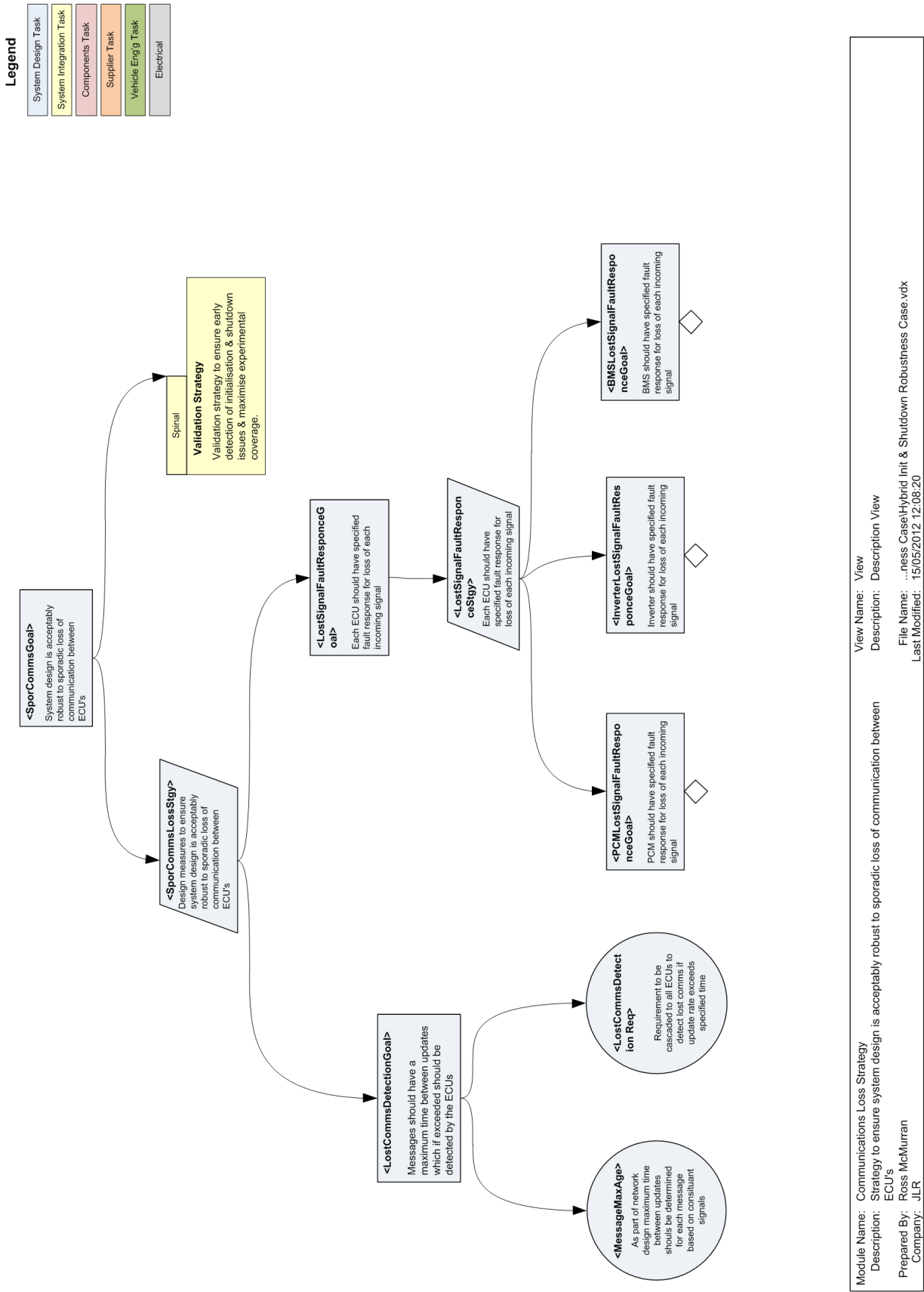


Hang-Ups

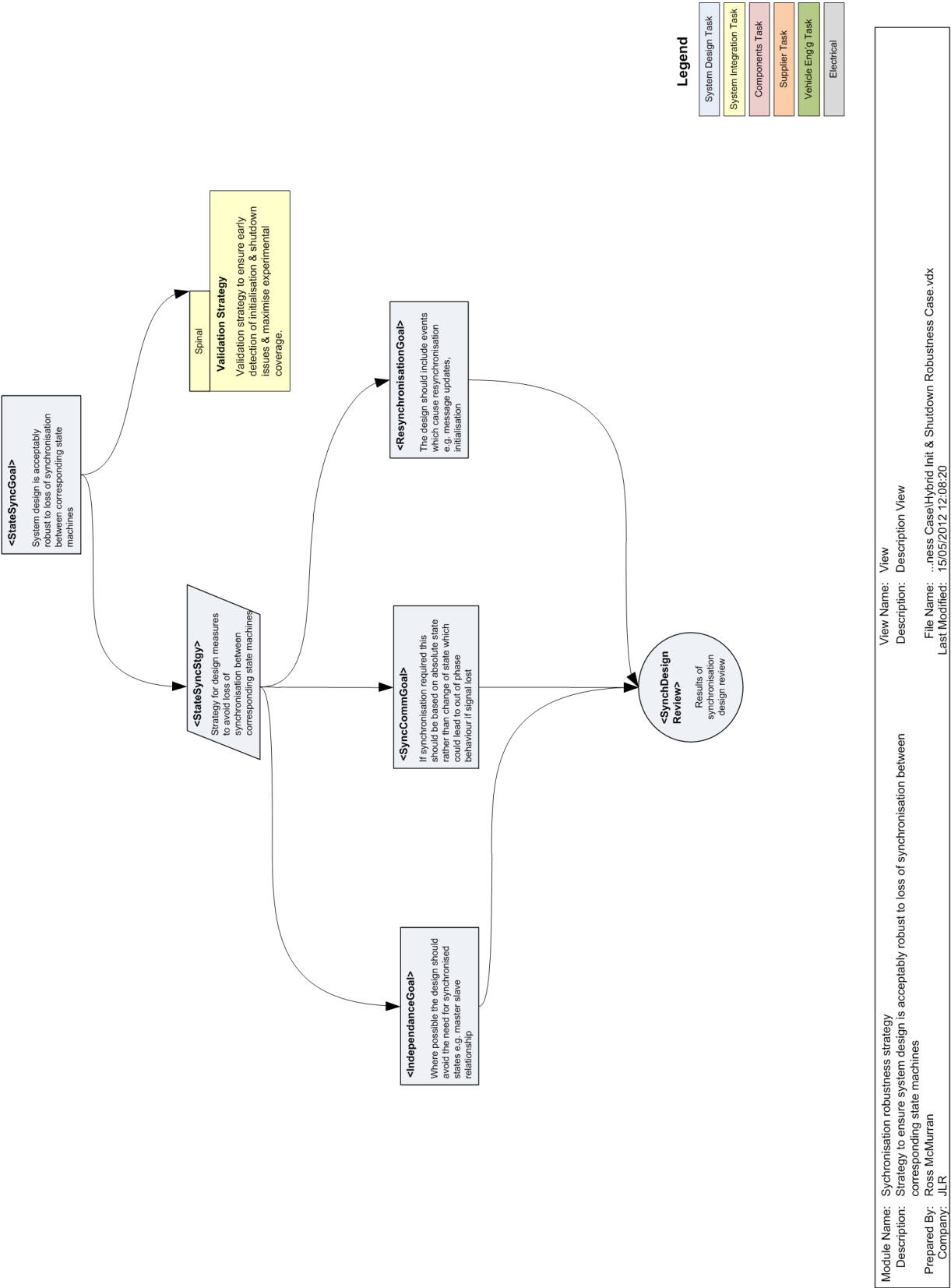


Hybrid System Initialisation Robustness Case – Argument for Robustness to Sporadic Loss of

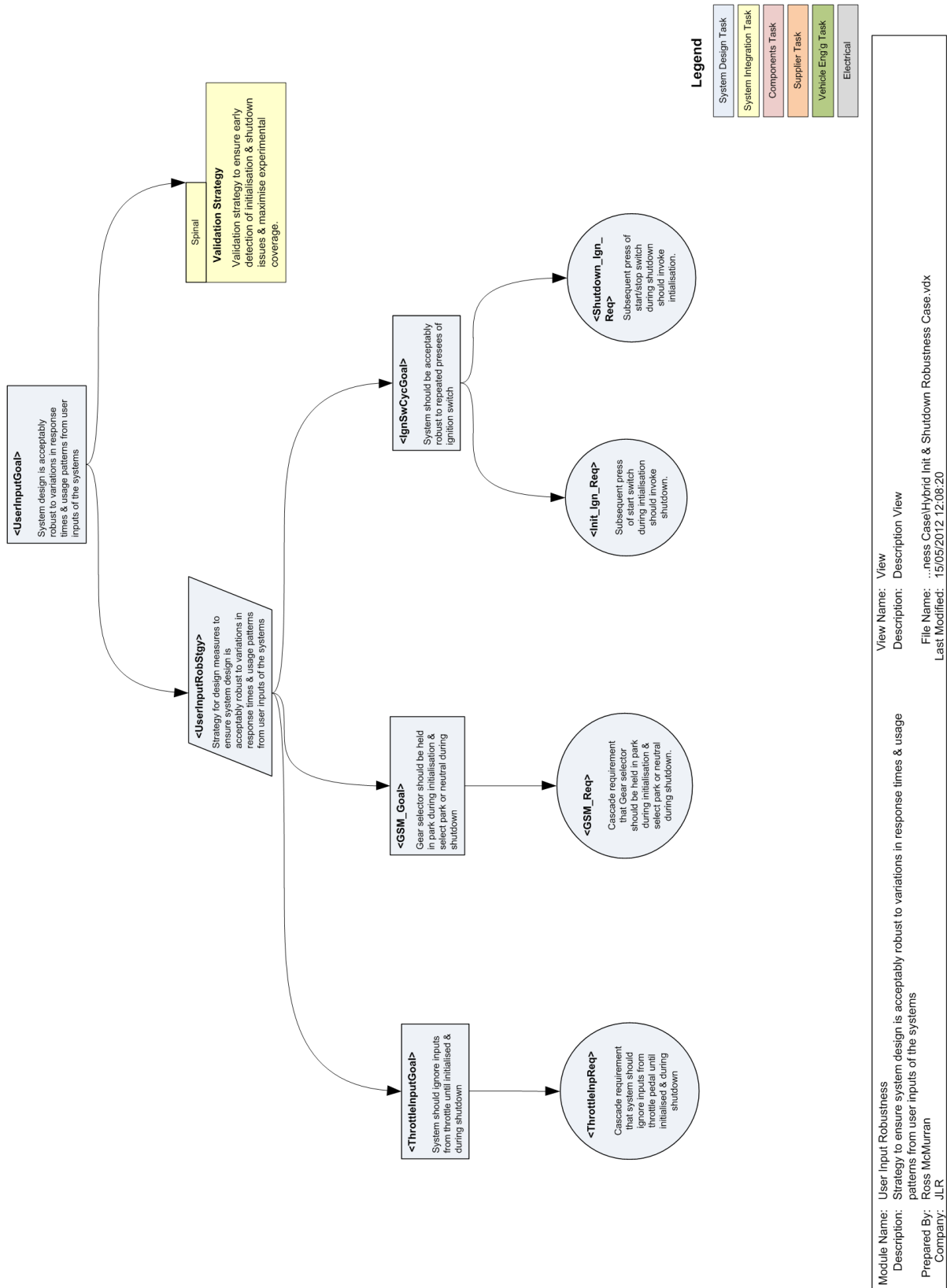
Communication



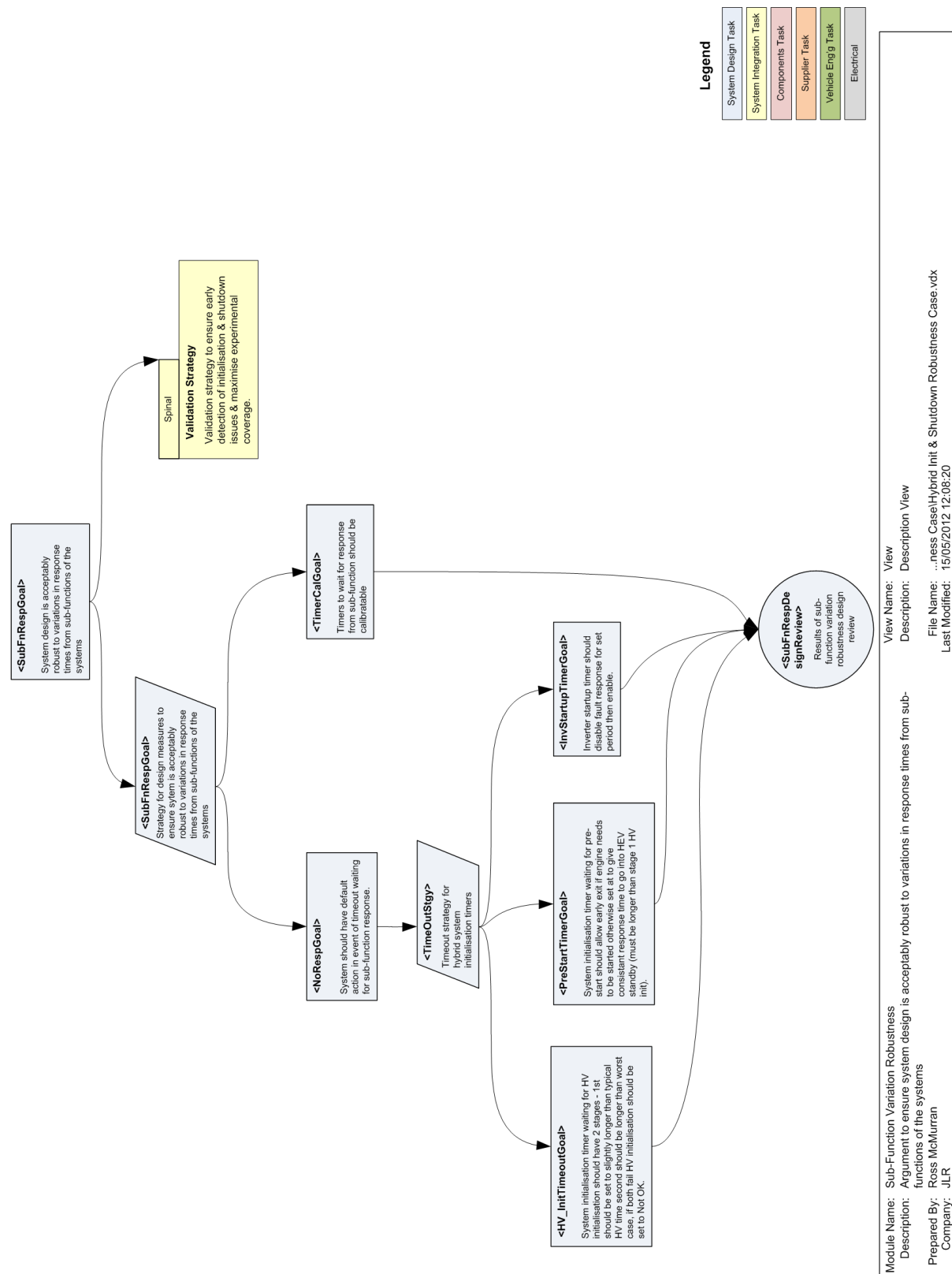
Hybrid System Initialisation Robustness Case – Argument for Robustness to Loss of Synchronisation Between Corresponding State Machines



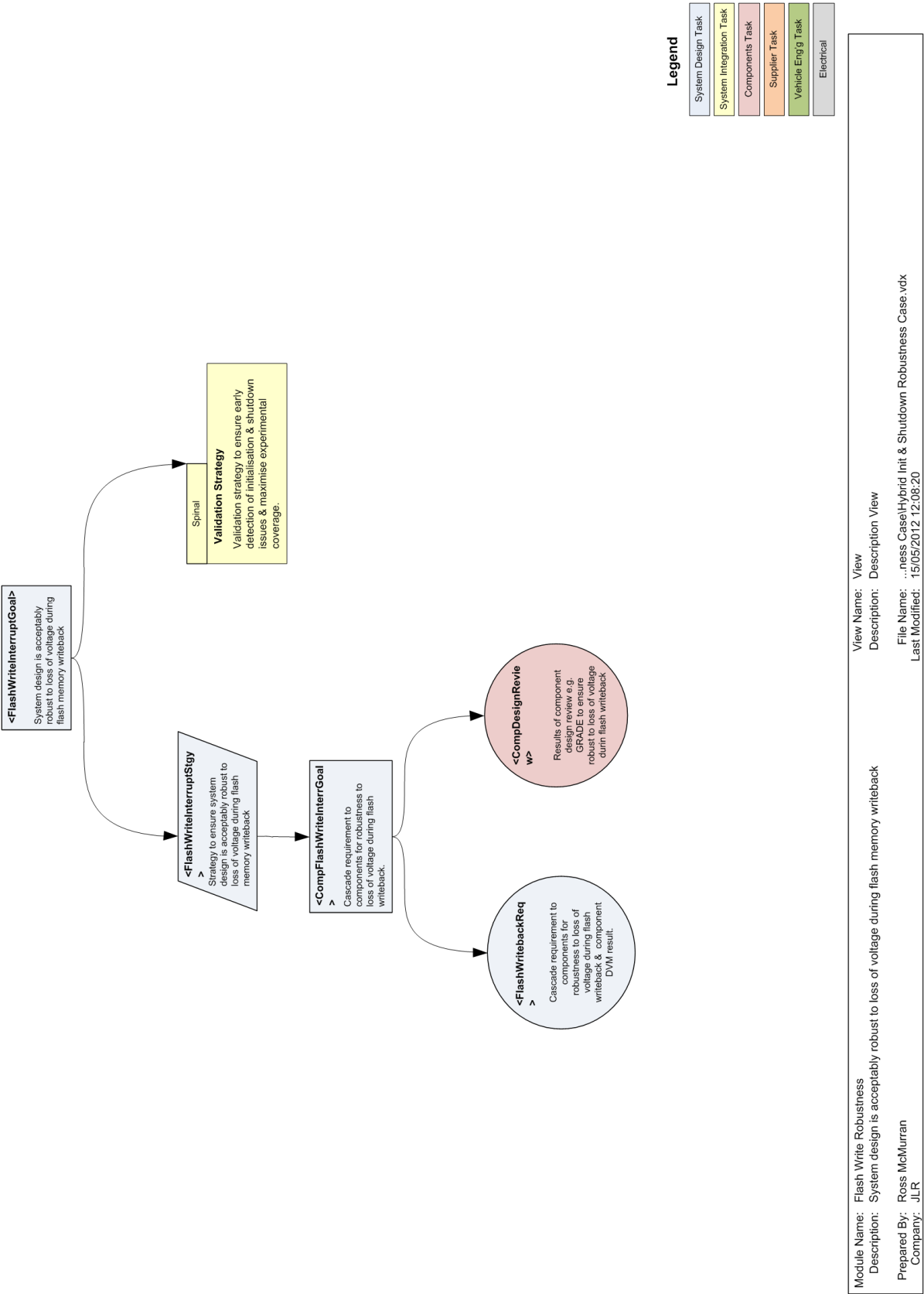
Hybrid System Initialisation Robustness Case – Argument for Robustness to User Inputs



Hybrid System Initialisation Robustness Case – Argument for Robustness to Variations in Response Times from Sub-Functions

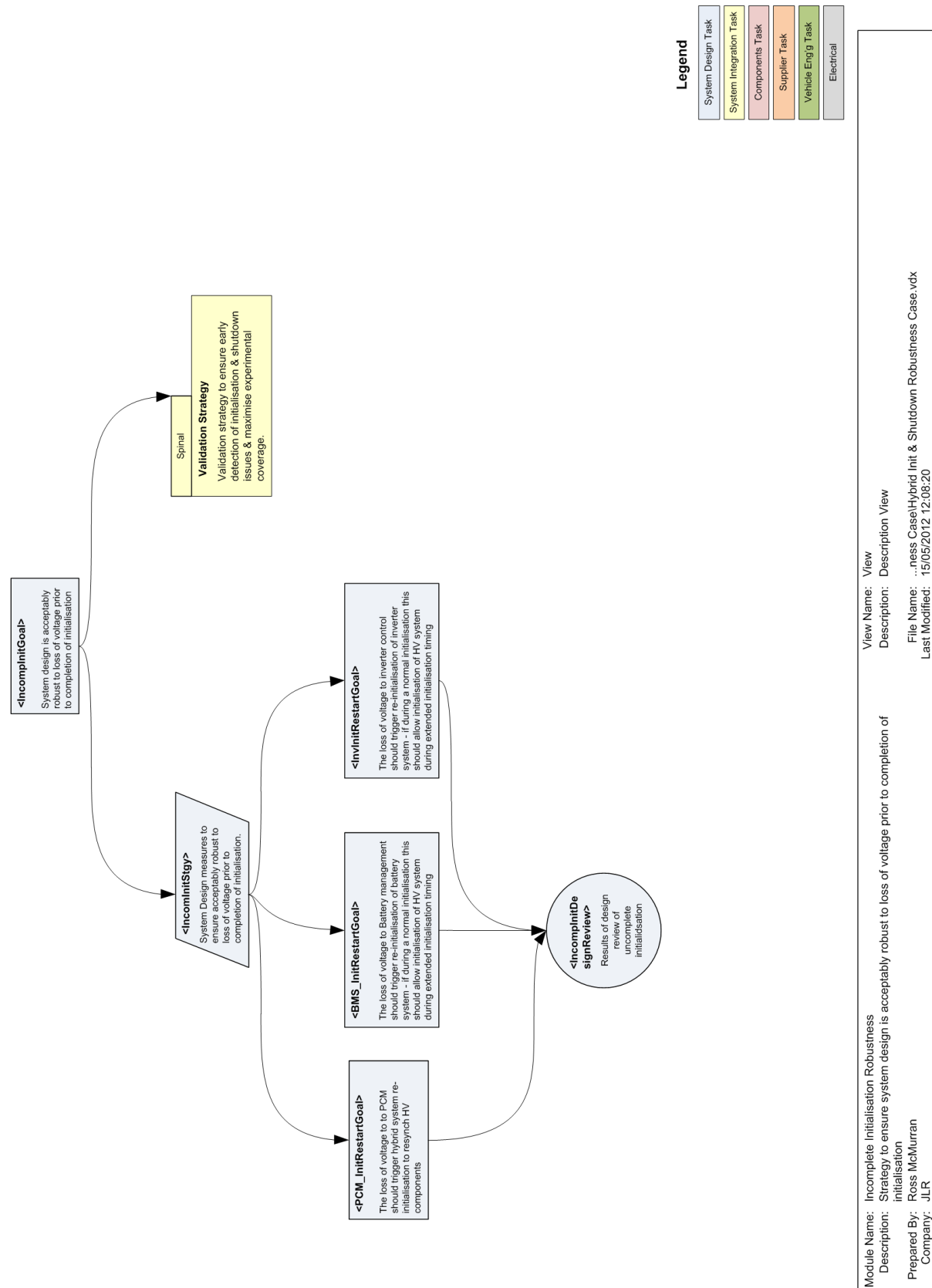


Hybrid System Initialisation Robustness Case – Argument for Robustness Due to Loss of Voltage During Flash Memory Writeback



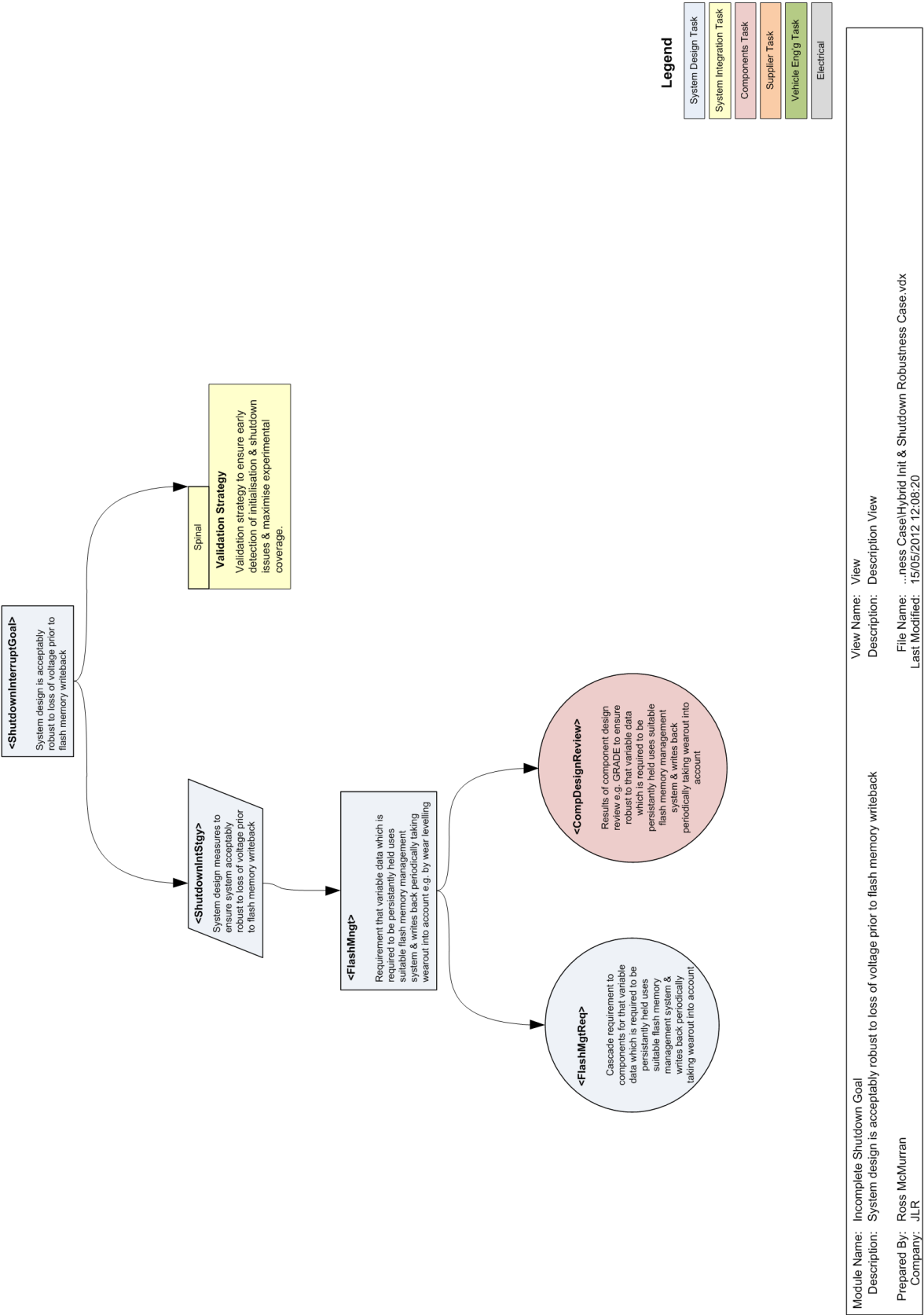
Hybrid System Initialisation Robustness Case – Argument for Robustness to Incomplete

Initialisation



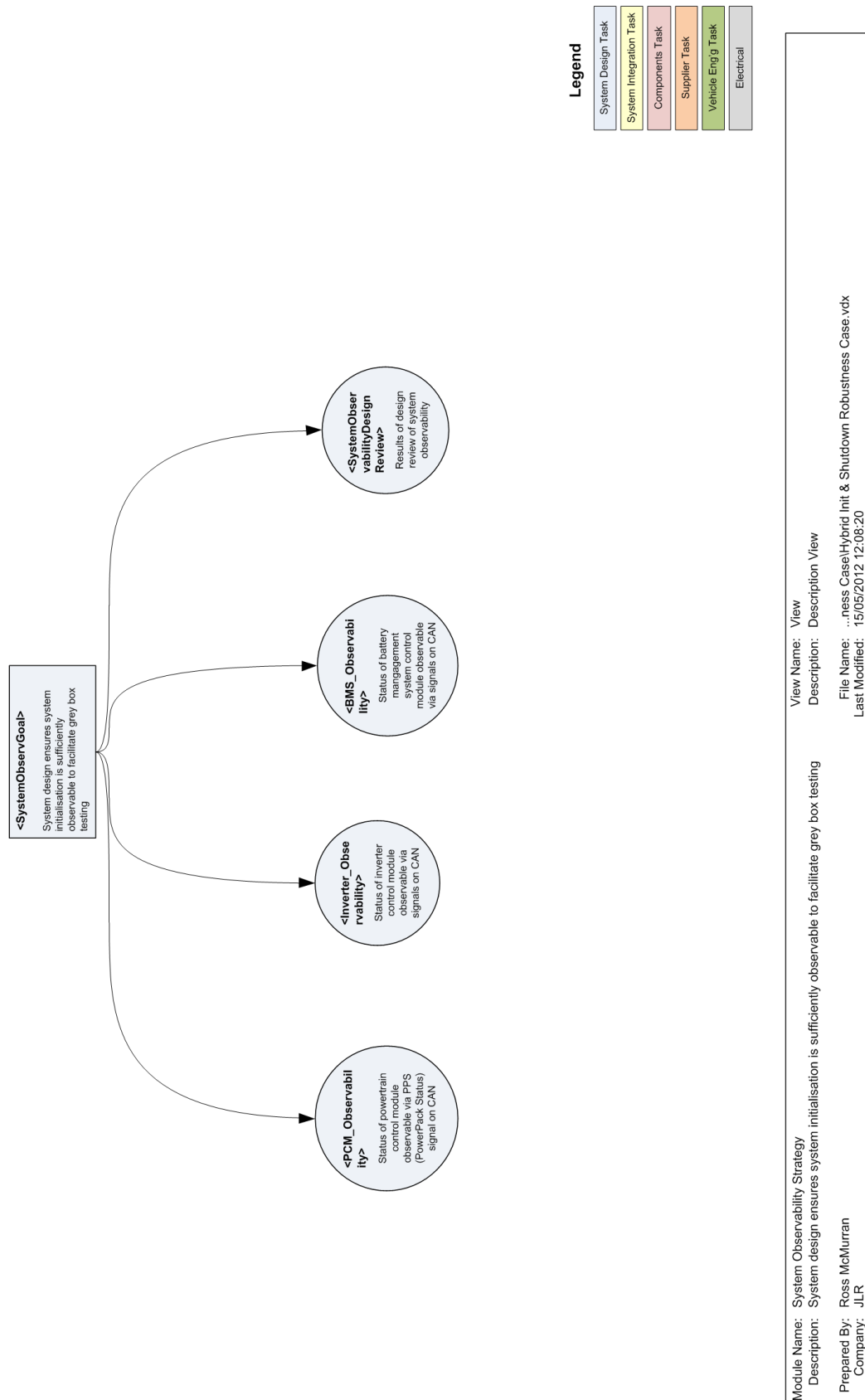
Hybrid System Initialisation Robustness Case – Argument for Robustness to Incomplete

Shutdown

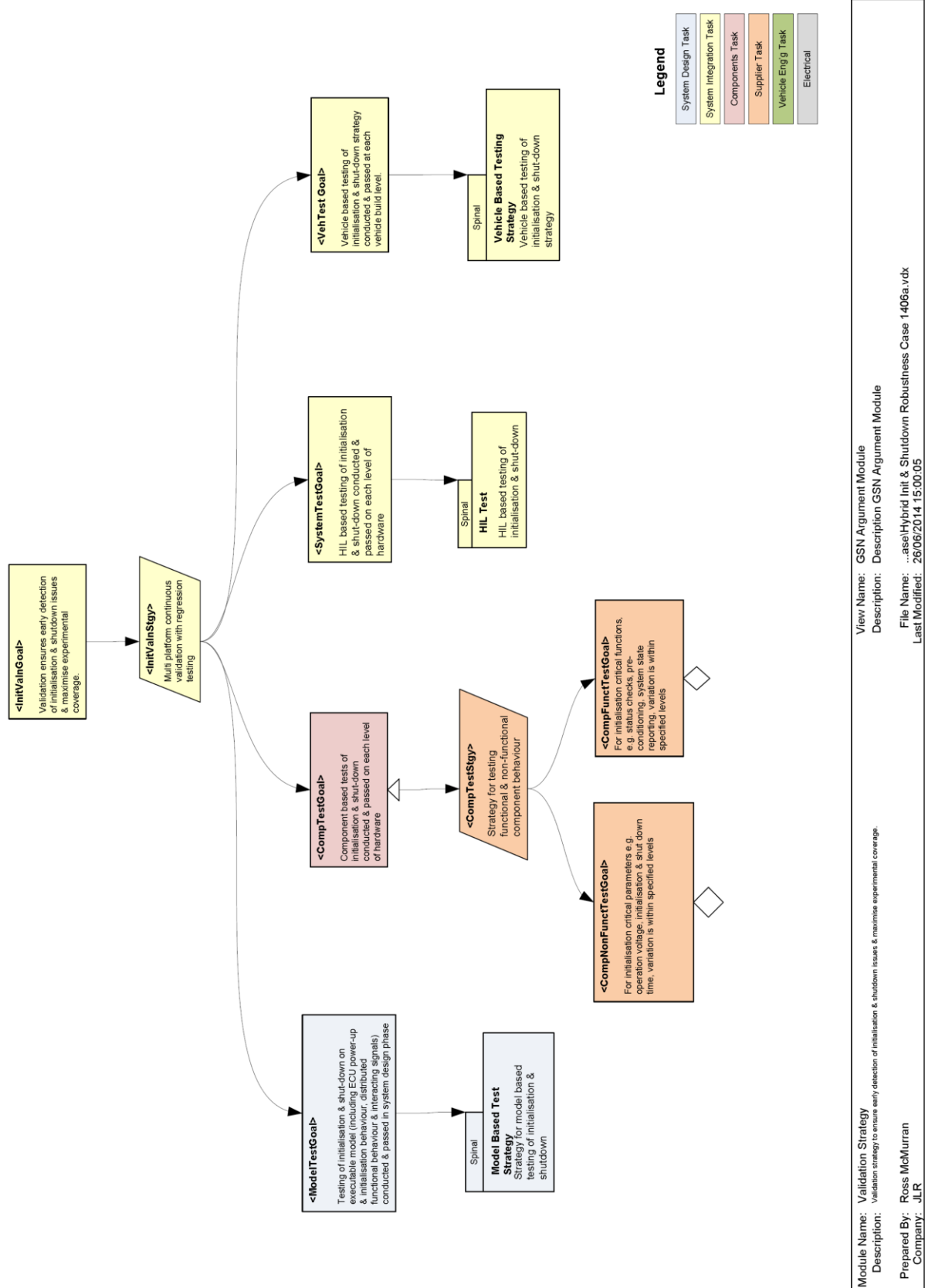


Hybrid System Initialisation Robustness Case – Argument for System Observability to

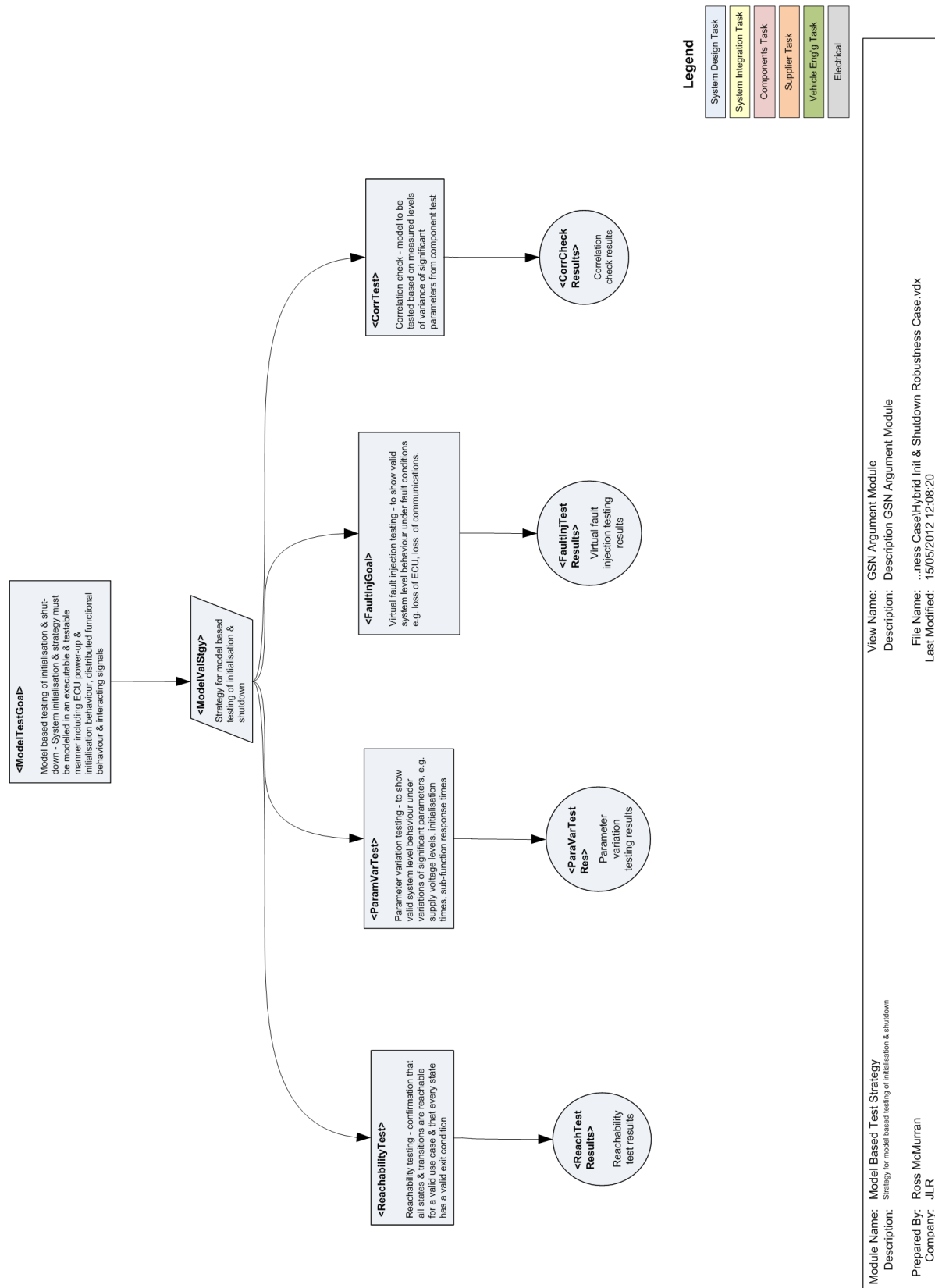
Support Grey Box Testing



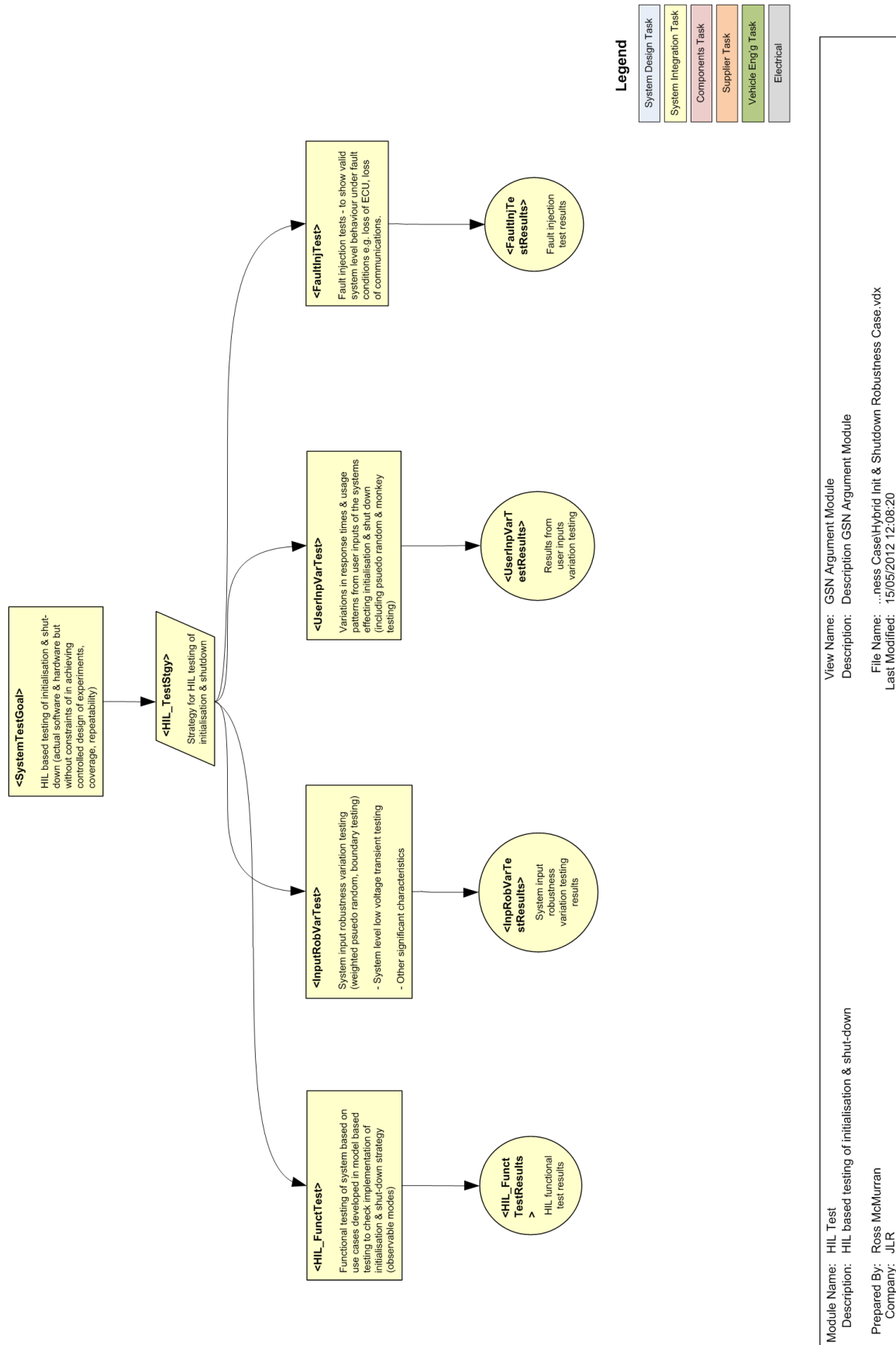
Hybrid System Initialisation Robustness Case – Argument for System Verification Approach



Hybrid System Initialisation Robustness Case – Argument for Model Based Test Strategy



Hybrid System Initialisation Robustness Case – Argument for HIL Testing



Hybrid System Initialisation Robustness Case – Argument for Vehicle Based Test Strategy

